TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

REIMA PIILILÄ

ATAM-MENETELMÄN SOVELTAMINEN PIENPROJEKTEISSA

Diplomityö

Tarkastaja: professori Hannu-Matti Järvinen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan tiedekuntaneuvoston kokouksessa
5. joulukuuta 2012

TIIVISTELMÄ

PIILILÄ, REIMA: ATAM-menetelmän soveltaminen pienprojekteissa
Tampereen teknillinen yliopisto
Diplomityö, 70 sivua, 0 liitesivua
Helmikuu 2016
Signaalinkäsittelyn ja tietoliikennetekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Sulautetut järjestelmät
Tarkastaja: professori Hannu-Matti Järvinen

Avainsanat: ATAM, arkkitehtuurianalyysi, ohjelmistoarkkitehtuuri, pienprojekti, soveltaminen

Ohjelmistoarkkitehtuurin analysointi on toimenpide, millä pyritään löytämään arkkitehtuurista riskejä, jotka voivat realisoitua joko kehitysvaiheessa tai valmiin ohjelmiston normaalissa käytössä. Riskien kartoittamisella pyritään saavuttamaan mm. säästöjä projektin kustannuksissa. Arkkitehtuurin analysointiin löytyy lukuisia eri menetelmiä, joista tämän työn puitteissa keskitytään vain ATAM-menetelmään. ATAM on muihin menetelmiin verrattuna raskas – pääosin miestyötuntien perusteella – ja sopii siten parhaiten isokokoisiin projekteihin. Tämän työn tarkoituksena on keventää ATAM-menetelmää siten, että se soveltuisi kustannuksiltaan myös pienprojektien käyttöön.

Tässä työssä on sovellettu ATAM-menetelmää arkkitehtuuriin, mikä on piirteiltään modulaarinen ja varioitava. Arkkitehtuurin pohjalta on ennen analyysiä julkaistu jo kaksi valmista tuotetta, jolloin ilmeisimmät puutteet ovat jo korjattu. Projektin kehitystiimin koko on supistunut kahteen aktiiviseen kehittäjään, mikä on ATAM-menetelmän laajuuden kannalta hyvin pieni projekti. Arkkitehtuurin riskien kartoituksen ohessa on pyritty kartoittamaan ATAM-menetelmän muunneltavuutta ja sen soveltuvuutta pienessä mittakaavassa.

Työssä on esitetty ATAM-menetelmän kaikki työvaiheet suppeasti, sekä muutokset, mitä menetelmään on tehty. Muunnelmassa on pyritty supistamaan menetelmää pienemmän projektin skaalaan siten, että analyysin arvioitu kustannustehokkuus ei kärsisi. Työstä käy ilmi, että menetelmään soveltaminen on osittain tilannekohtaista, mikä riippuu paljolti käytettävissä olevista resursseista, resurssien tietotaidosta ja menetelmän keventämistarpeesta. Käytäntöön soveltaminen osoitti, että menetelmä on suhteellisen helposti kevennettävissä, mutta pienprojektin ja arkkitehtuurin tila vaikuttavat paljon analyysin luonteeseen ja siitä saataviin tuloksiin. Valmiin tuotteen analysointi poikkeaa suuresti tekemättömän arkkitehtuurin analyysistä.

ABSTRACT

PIILILÄ, REIMA: Application of ATAM-method in small scale projects.
Tampere university of technology
Master of Science Thesis, 70 pages, 0 Appendix pages
February 2016
Master's Degree programme in Signal Processing and Communications Engineering
Major: Embedded Systems
Examiner: Professor Hannu-Matti Järvinen

Keywords: ATAM, architecture analysis, software architecture, small scale project, modification

Software architecture analysis is a task, which is used to find risks from the architecture. These risks might become real obstacles during development or during normal operation of a ready product. The reason why risks are tried to be found is to get savings in overall costs of the project. There are multiple methods for architecture analysis however this thesis concentrates only on ATAM-method. ATAM is quite laborious compared to other methods and is therefore best suited for large scale projects. The purpose of this thesis is to modify ATAM-method into a bit lighter form so that it would be better suited for small scale projects.

ATAM-method is used in this thesis for analyzing an architecture which is both modular and which can be varied into multiple different products. Before the analysis was done, two different products had been developed from this architecture. Most of the problems in the architecture had been solved during that time. The active development team had been reduced to only two members, which makes this quite a small project compared to the regular size of the projects analyzed with ATAM-method. While trying to map out risks from the architecture, ATAM-method was being evaluated for its modifiability and suitability for small scale projects.

This thesis goes through all tasks and phases of ATAM-method briefly and the modifications done to the method. The modification tries to reduce the amount of work needed for full evaluation so that it suits better for small scale projects. Cost-efficiency is also taken into consideration while these modifications are done. It becomes clear from this thesis that modification must be done based on available resources and people's skills and how much the costs need to be reduced. Practical application showed that this method is relatively easy to modify however the state of the project and architecture affects greatly on the results of the analysis. There is a great difference in analyzing an already finished product compared to an unfinished one.

ALKUSANAT

Tämä diplomityö on tehty Symbiolle vuosina 2012 – 2016. Työssä tarkastellaan ATAM-menetelmän muunneltavuutta pienprojektin käyttöön ja käytetään muunneltua menetelmää valmiin arkkitehtuurin analysoimisessa.

Haluan kiittää työkavereitani, jotka osallistuvat arkkitehtuurin evaluointiin. Ilman heitä tätä diplomityötä ei olisi voinut tehdä. Kiitos myös kollegoilleni Ari Hyttiselle ja Kai Tusalle oikolukemisesta ja kannustavista puheista. Haluan kiittää myös professori Hannu-Matti Järvistä tämän työn aiheen keksimisestä sekä asiantuntevasta ohjauksesta.

Kiitos myös sukulaisille pitkämielisyydestä ja kannustamisesta.

Tampere, 15.2.2016

SISÄLLYS

1 Johdanto 1
2 Arkkitehtuurien analysointi ja ATAM-menetelmä 2
2.1 Arkkitehtuurien analysointi 2
2.2 ATAM analyysin suoritus 3
2.2.1 Analyysiin osallistujat ja niiden roolit 3
2.2.2 Laatuominaisuudet, laatupuu ja skenaariot 5
2.2.3 Herkkyyskohdat, tasapainokohdat ja riskit 7
2.2.4 Analyysin toimenpiteet 8
2.2.5 Analyysin jako neljään vaiheeseen 11
2.3 ATAM-menetelmän tulokset ja sivutuotteet 13
2.4 Analysoinnin edut ja kustannukset 14
2.5 Muut evaluointimenetelmät 15
2.6 Yhteenveto 16
3 Analysoitava ohjelmisto 17
3.1 Analysoitava ohjelmisto ja sen tehtävät 17
3.2 Arkkitehtuurisuunnittelua ohjaavat vaatimukset ja ominaisuudet 18
3.2.1 Yleiset vaatimukset ja laatuominaisuudet 18
3.2.2 Liiketoimintanäkökulmat 19
3.3 Arkkitehtuurin kuvaus 20
3.3.1 Yleiskuvaus 21
3.3.2 Laitetason näkymä 22
3.3.3 Moduulit 24
3.3.4 Arkkitehtuuri kokonaisuutena 27
3.3.5 Vahtikoira 30
3.3.6 Datan virtaus arkkitehtuurissa 31
3.4 Varioinnin vaikutus arkkitehtuuriin 33
3.5 Arkkitehtuurin tuomat haasteet 35
4 ATAM-menetelmän soveltaminen ja tulokset 37
4.1 ATAM-menetelmän modifikaatio 37
4.2 Työn kulku 38
4.2.1 Suoritus 38
4.2.2 Yleiset haasteet 40
4.2.3 Arvio onnistumisesta 41
4.3 Laatupuu 42
4.4 Skenaariot 43
4.4.1 Skenaario 1 – Käyttöliittymän datojen peilaus 44
4.4.2 Skenaario 2 – Viestitulva CAN-väylästä 49
4.5 Riskit ja riskiteemat 54
5 Arviot ja jatkokehitys 55
5.1 Ohjelmiston tila vuosi analyysin jälkeen 55

5.2 Arkkitehtuurin sopivuus vaadittuun tehtävään 56
5.3 ATAM-menetelmän arviointi 57
5.4 Hyödyn suhde kustannuksiin 58
6 Yhteenveto 60
Lähteet 62

TERMIT JA NIIDEN MÄÄRITELMÄT

Application layer	Analysoitavan arkkitehtuurin keskimmäinen kerros, ohjelmalogiikkakerros
ATAM	Architecture Tradeoff Analysis Method, arkkitehtuurin evaluointimenetelmä.
AT&T	Amerikkalainen televiestintäyritys
BitBake	Käännösohjelma, mikä kääntää käännösreseptien perusteella ohjelman ja asennettavan Linux-käyttöjärjestelmän. Soveltuu erityisesti hyvin pienikokoisten käyttöjärjestelmien kääntämiseen.
CAN	Controller Area Network. Erityisesti ajoneuvoissa ja teollisuudessa paljon käytetty dataväylä.
Data Provider layer	Analysoitavan arkkitehtuurin alin kerros, tietolähdekerros
Debian	Linux-jakelu, minkä päälle moni tunnettu Linux-jakelu on rakennettu.
ISO	International Organization for Standardization, standardisoimisjärjestö
Konfigurointi	Ohjelman toiminnan määrittely koodin ulkopuolisilla konfiguraatiotiedostoilla.
MySQL	Laajasti käytetty palvelimellinen transaktiotietokanta.
OSI-malli	Open Systems Interconnection Reference Model, tiedonsiirtoprotokollien yhdistelmää kuvaava malli.
Presentation layer	Analysoitavan arkkitehtuurin ylin kerros, käyttöliittymäkerros.
QAW	Quality Attribute Workshop, ATAM-menetelmää tukeva analysointimenetelmä.
QML	Qt Meta Language tai Qt Modeling Language. Deklaratiivinen käyttöliittymän määrittelykieli.
Rugged Computer	Tavallista kestävämmällä rakenteella suunniteltu tietokone.
SEI	Carnegie Mellon University, Software Engineering Institute. ATAM-menetelmän kehittänyt instituutti.
SAE J1939	CAN-väylästandardi, mitä käytetään laajasti autoteollisuudessa.
SQLite	Avoimena lähdekoodina levytyksessä oleva kevyt palvelimeton transaktiotietokanta, jonka tekijänoikeuksista on luovuttu.
UI	User Interface, käyttöliittymä.
Variointi	Uuden toisista tuotteista eroavan tuotekonfiguraation tekeminen käyttäen tuotteille yhteisiä ydinkomponentteja. Varioidussa tuotteissa voi olla toisistaan poikkeavia moduuleja ja moduulien konfigurointi voi erota.

XML	Extensible Markup Language, tallennusformaatti.

1 JOHDANTO

Ohjelmistojen noudattaa usein kaavaa, missä määritellään ja suunnitellaan, toteutetaan ja korjataan toteutusta. Joskus tuote voi valmistua pienelläkin työllä, mutta toisinaan ohjelmiston kehityksen, testauksen tai käyttöönoton myötä ohjelmiston korjaustarve voi olla kustannuksiltaan huomattavan suuri. Tuotekehityksen vaiheisiin on kehitetty useita arkkitehtuurin analyysitekniikoita, millä pyritään löytämään riskejä ennen niiden realisoitumista ja siten ohjaamaan tuotekehitystä parempaan suuntaan.

Tämän diplomityön puitteissa tutkitaan yhden arkkitehtuurien analysointitekniikan – ATAM-menetelmän – toimivuutta pienikokoisen projektin puitteissa. ATAM-menetelmä on ohjeelliselta rakenteeltaan raskas, eikä sen täysimittainen suoritus ole kustannustehokasta hyvin pienikokoisten projektien tapauksessa. Luvussa 2 esitellään ATAM-menetelmä sellaisena, miten se on tarkoitettu suoritettavan arkkitehtuurien analyysissä. Myöhemmin luvussa 4 esitellään modifioitu version ATAM-menetelmästä, missä on pyritty supistamaan kustannuksia menettämättä liikaa menetelmän tulosten tarkkuutta. Tätä on pyritty saavuttamaan supistamalla tai poistamalla hyötysuhteeltaan heikompia työvaiheita ja analyysiin osallistujia, samalla kun tärkeimmät työvaiheet on pyritty säilyttämään ennallaan.

Luvussa 3 on esitelty arkkitehtuuri sillä tarkkuudella, mikä on nähty luvun 4 (ATAM-menetelmän soveltaminen ja tulokset) ymmärtämisen kannalta tarpeelliseksi ja riittäväksi. Kuvattua arkkitehtuuria on kehitetty kohtuullisen paljon, joten se ei vastaa aivan pienimpien projektien kokoluokkaa. Arkkitehtuuriin liittyy paljon yksityiskohtia ja tarkempi kuvaus vaatisi huomattavasti enemmän dokumentointia, kuin mitä tähän diplomityöhön on mahdollista sisällyttää. Siitä johtuen arkkitehtuuriluvusta on supistettu tämän diplomityön kannalta epäoleelliset asiat pois.

Ensimmäiset luvut, luvut 2-4 on kirjoitettu samalla, kuin arkkitehtuurin analyysiä on tehty. Luku 5 (Arviot ja jatkokehitys) on kirjoitettu vuosi työvaiheen jälkeen, jolloin analyysin perusteella tehdyt jatkokehityshankkeet ovat jo osittain realisoituneet. Vuoden tauon myötä analyysin tuloksien ja siitä saatuja hyötyjen tarkastelu on hieman realistisempaa, sillä tuotteen elinkaaren ollessa pitkä, analyysin tulokset voivat vaikuttaa pitkälle tulevaisuuteen. Luvun 5 sisältö kattaa arkkitehtuurille tehdyt jatkotoimenpiteet ja arkkitehtuurin arvioinnin, sekä ATAM-menetelmän arvioinnin ja kustannuksista saatuja hyötyjä.

2 ARKKITEHTUURIEN ANALYSOINTI JA ATAM-MENETELMÄ

Tässä luvussa käsitellään arkkitehtuurien analysointia, niihin liittyviä menetelmiä, etuja ja kustannuksia. Pääpainona on kuitenkin käydä läpi ATAM-menetelmän teoria, joka on käsitelty aliluvussa 2.2. Teoria on esitetty hyvin kompaktisti suhteessa siihen, että ATAM-menetelmän kehittänyt SEI (Carnegie Mellon University, Software Engineering Institute) on kirjoittanut aiheesta useamman kirjan ja lukuisia julkaisuja. Muissa luvuissa käsitellään evaluointia hieman yleisemmin.

2.1 Arkkitehtuurien analysointi

Ennen kuin arkkitehtuuria voidaan analysoida on selvitettävä, mitä ohjelmistorakkitehtuurilla ylipäätään tarkoitetaan. Arkkitehtuurille löytyy eri lähteistä kymmeniä toisistaan poikkeavia määritelmiä, mikä osoittaa, että arkkitehtuurin määritelmä ei ole täysin yksiselitteinen [1]. Yhteistä määritelmillä on kuitenkin se, että arkkitehtuuri on järjestelmän rakenteen sekä toiminnan määritelmä ja se pyritään kuvaamaan eri lähestymistavoilla. Arkkitehtuuri siis kuvaa, mistä osista järjestelmä koostuu, mitkä ovat niiden suhteet ja miten osat toimivat keskenään. Arkkitehtuuri kattaa sekä staattiset käännösaikaiset rakenteet että ajoaikaiset dynaamiset rakenteet. [4]

Koska arkkitehtuuri on järjestelmän tai järjestelmien abstraktio, jonka määritelmät ohjaavat toteutusta, sen merkitys on lopputuotteen onnistumisen kannalta erittäin suuri. Onnistumiseen vaaditaan, että arkkitehtuuri vastaa sille asetettuihin rajoitteisiin ja vaatimuksiin ja että se on ylipäätään toteutettavissa. Järjestelmää suunnitellessa kaikki vaatimukset eivät välttämättä ole tulleet vielä julki, sen lisäksi valituissa suunnitteluratkaisuissa voi piillä riskejä, joita ei ole otettu huomioon. Mikäli riskejä ja määrittelemättömiä vaatimuksia ei löydetä projektin alkuvaiheessa, tarvittavien korjaavien muutosten tekeminen voi tulla projektin loppuvaiheessa kohtuuttoman kalliiksi. Näiden riskien ja vaatimusten kartoittamiseen on kehitetty useita eri analysointimetodeja, josta yksi on ATAM. Mikäli analyysi tehdään hyvin varhaisessa vaiheessa, arkkitehtuurillisilla muutoksilla voidaan välttyä ainakin osalta analyysissä ilmenneiltä riskeiltä. Toisaalta analyysillä saadaan selville voiko esitetyllä arkkitehtuurilla ylipäätään saavuttaa sille asetettuja laatuvaatimuksia. [1]

Arkkitehtuurin voi analysoida missä vaiheessa tahansa, kunhan järjestelmästä on olemassa riittävän tarkka kuvaus. Analyysi ajoitetaan useimmiten joko projektin alkuvaiheeseen, jolloin arkkitehtuuriin voi vielä tehdä muutoksia, tai implementoinnin päättymisen jälkeen, jolloin analyysin tuloksia voidaan hyödyntää järjestelmän jatkokehityksessä. Hyödyllisin hetki suorittaa analyysi on tietysti projektin alussa siinä vaiheessa, kun arkkitehtuuri on suunniteltu, mahdollisesti ennen kuin vaatimukset on

jäädytetty. Alkuvaiheessa tehdyn analyysin tuloksena voi parhaimmassa tapauksessa tehdä muutokset sekä vaatimuksiin että arkkitehtuuriin siten, että hankalimmat yllätykset saadaan karsittua pois. [1]

Analyysin hyödyt eivät rajoitu pelkästään riskinhallinnallisiin seikkoihin, vaan analyysimetodista riippuen prosessin suorittaminen tuottaa usein sivutuotteita. Evaluoinnin esivaatimukset pakottavat arkkitehtiä selventämään arkkitehtuurin kuvausta ja parantamaan dokumentaation laatua. Prosessin aikana vuorovaikutus asiakkaan kanssa paranee ja vaatimukset selkenevät. Lisäksi tieto leviää organisaation sisällä ja voi tulla ilmi komponenttien uudelleenkäytettävyysmahdollisuuksia projektien välillä. ATAM-mallin mukaisen analyysin lopputuotteet käsitellään aliluvussa 2.3. [1]

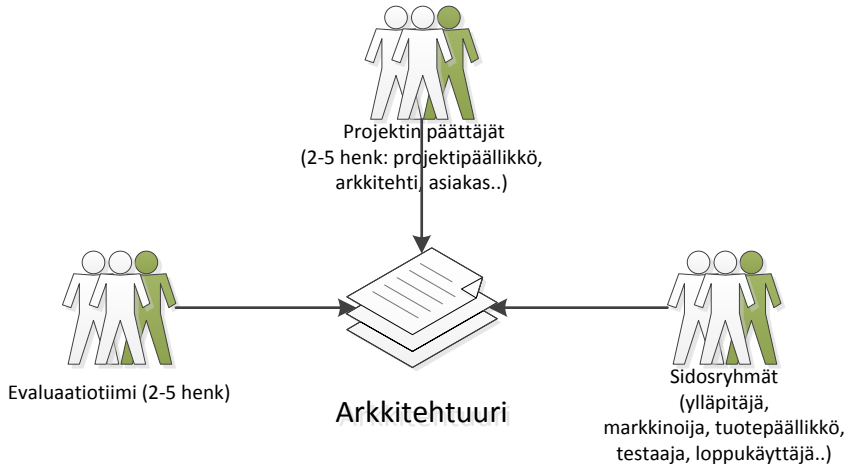
On ilmeistä, että järjestelmän analysointi ei ole ilmaista, vaan se vaatii jonkinlaisen ajallisen panostuksen. Menetelmästä riippuen evaluointi voi olla esimerkiksi lyhyt tarkistuslistan läpikäynti, kokeellinen prototyyppiarkkitehtuurin testaus tai isohko monta ihmista yhteen kokoava prosessi. ATAM lukeutuu näistä viimeiseen, ollen siten oppikirjanmukaisesti suoritettuna suhteellisen kallis menetelmä. Menetelmistä voi yleisesti sanoa sen, että kevyellä menetelmällä saa löydettyä ja siivottua kaikkein ilmeisimmät riskit, kun taas vaikeammin löydettävien, potentiaalisesti yhtä suurien riskien löytämiseen tarvitaan perusteellisempaa menetelmää. ATAM-menetelmä pyrkii perusteelliseen evaluointiin ottamalla mukaan näkemyksiä mahdollisimman monipuolisesti, mutta samaan aikaan olemalla kompakti prosessi, joka ei vie osallistujilta paljon kalenteriaikaa.

2.2 ATAM analyysin suoritus

ATAM-pohjainen arkkitehtuurin analyysi on kuvattu kirjallisuudessa hyvin tarkkaan ja siitä on olemassa paljon materiaalia. Tässä aliluvussa käy ilmi, että ATAM on luonteeltaan skenaariopohjainen ja analyysissä lähestytään arkkitehtuuria laatuvaatimuksien näkökulmasta. Tämän luvun sisällössä käsitellään analyysiin tarvittavat osallistujat (aliluku 2.2.1), evaluoinnin työvaiheita tukeva pohjateoria (aliluku 2.2.2), evaluoinnin jako toimenpiteisiin (aliluku 2.2.4) ja toimenpiteiden jako vaiheisiin (aliluku 2.2.5).

2.2.1 Analyysiin osallistujat ja niiden roolit

ATAM-analyysiin osallistuvat ryhmät ja niiden vastuut evaluoinnissa on kirjallisuudessa melko tarkkaan määritelty, mutta ryhmien tarkempi kokoonpano on jätetty analyysin tekijöiden päätettäväksi. Osallistujat jakautuvat karkeasti kolmeen ryhmään, joista ensimmäinen on analyysiä johtava ja prosessin kulkua tarkkaileva evaluointitiimi. Toinen ryhmä koostuu projektin päättäjistä, jotka tuntevat järjestelmän ja joilla on valta tehdä arkkitehtuurillisia päätöksiä. Viimeinen ryhmä koostuu muista sidosryhmistä, joihin lukeutuvat mm. loppukäyttäjä tai ylläpitäjä. He tuovat omat käytännönläheiset näkemyksensä järjestelmälle asetettavista vaatimuksista. Kuva 1 esittelee evaluointiin osallistuvat ryhmät ja niiden ohjeelliset koot.



Kuva 1 Evaluointiin osallistuvat ryhmät

Evaluointitiimin merkitystä analyysissä ei voi korostaa liikaa. Tiimin harteilla on ohjata analyysi ATAM-mallin määrittelemän prosessin mukaisesti ja tuoda mukaan oma laaja arkkitehtuurillinen kokemuksensa, kun arkkitehtuurillisia suunnitteluratkaisuja analysoidaan. Kirjallisuus esittää tiimin jäsenille tarkkaa roolijakoa, jossa jäsenille annetaan tietyt ennaltamääritellyt vastuut analyysin kulun ajaksi. Näitä vastuita on tiiminjohtaja, evaluoinninjohtaja, kirjuri, ajan tarkkailija, prosessin tarkkailija, prosessiin pakottaja ja kyselijä. Kukin evaluoija voi saada useamman vastuun samanaikaisesti ja toisaalta useampi evaluoija voi jakaa samoja vastuita keskenään. Evaluoinnin onnistumisen kannalta tärkeintä on kuitenkin se, että evaluoijat ovat kokeneita arkkitehtejä ja osaavat tunnistaa arkkitehtuurin suunnitteluratkaisuista niissä piilevät riskit. [2]

Projektin päättäjät on ryhmä, joka vastaa järjestelmästä ja jolla on valta tehdä arkkitehtuurillisia päätöksiä. Tärkeimmät osakkaat ryhmässä ovat projektipäällikkö, arkkitehti ja asiakas. Projektin päättäjät tuntevat järjestelmän ja siten osaavat sekä esitellä sen että vastata evaluointitiimin tekemiin kysymyksiin. Arkkitehdin läsnäolo on kaikkein tärkein, sillä järjestelmää ja sen arkkitehtuuria tulee selittää koko prosessin keston ajan. Asiakkaan tärkeimpiin tehtäviin lukeutuu järjestelmälle asetettujen vaatimuksien selvittäminen ja tarvittaessa ristiriitaisten vaatimusten muuttaminen. [2]

Viimeisenä ryhmänä analyysiin kutsutaan muut sidosryhmät, joilla on intressejä arkkitehtuurin suhteen ja joiden läsnäolo nähdään tarpeellisena. Sidosryhmiin lasketaan kaikki, jotka liittyvät jollain tavalla järjestelmään ja siten tuovat henkilökohtaisen näkemyksensä, mitä he vaativat järjestelmältä. Sidosryhmistä mm. loppukäyttäjä tuo näkemyksensä, miten hän järjestelmää käyttäisi, kun taas ylläpitäjä tuo näkemyksensä ylläpitoseikoista. Sidosryhmät osallistuvat analyysiin vasta loppuvaiheessa ja prosessin puolivälissä tehdään päätös, ketkä sidosryhmistä kutsutaan analyysiin mukaan.

Analyysiin osallistujien kokoonpano vaihtelee analyysin vaiheesta riippuen ja siitä, millä mittakaavalla analyysi halutaan suoritettavan. Pienimmillään analyysi suoritetaan muutaman henkilön voimin, kun taas suuremman mittakaavan analyysiin osallistuu yli kymmenen henkilöä. Tärkeintä on kuitenkin saada pöydän ympärille ne henkilöt, joilla on analyysiin eniten annettavaa ja joiden avulla analyysiin käytetystä ajasta saadaan

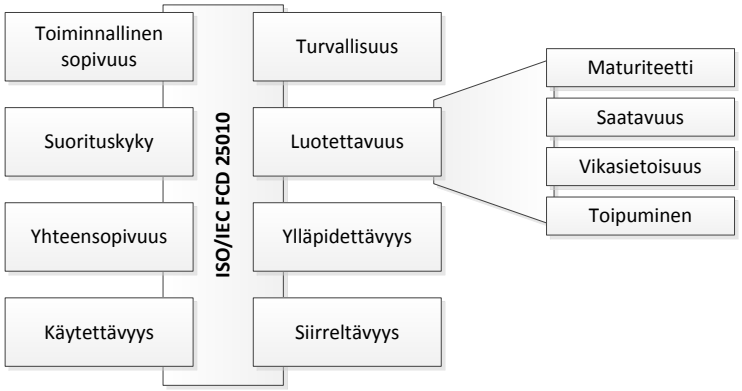
eniten irti. Mikäli analyysiin ei saada oikeita henkilöitä tai ryhmät jäävät liian pieniksi, on riski, että arkkitehtuurista jää huomaamatta vakavia riskitekijöitä. Taulukko 1 esittää evaluointiin osallistujien ajallisen käytön rakenteen täysimittaisessa ATAM-menetelmässä. Taulukossa mainitut vaiheet käsitellään aliluvussa 2.2.5. [2]

Taulukko 1 Täysimittaisen ATAM-menetelmän osallistujien ajalliset panostukset [1]

Vaihe	Evaluointitiimi (oletetaan 5 henkilöä)	Projektin päättäjät (arkkitehti, asiakas ja projektipällikkö)	Muut sidosryhmät (oletetaan 8 henkilöä)
Vaihe 0: valmistelu	1 päivä (tiiminjohtaja)	1 päivä	0
Vaihe 1: ensimmäinen evaluointi (1 päivä)	5 päivää	3 päivää	0
Vaihe 2: Täysi evaluointi (3 päivää)	15 päivää	9 päivää + 2 päivää valmisteluun	16 päivää
Vaihe 3: Loppuvaihe	15 päivää	3 päivää lukea ja vastata raporttiin	0
Summa	36 miestyöpäivää	18 miestyöpäivää	16 miestyöpäivää

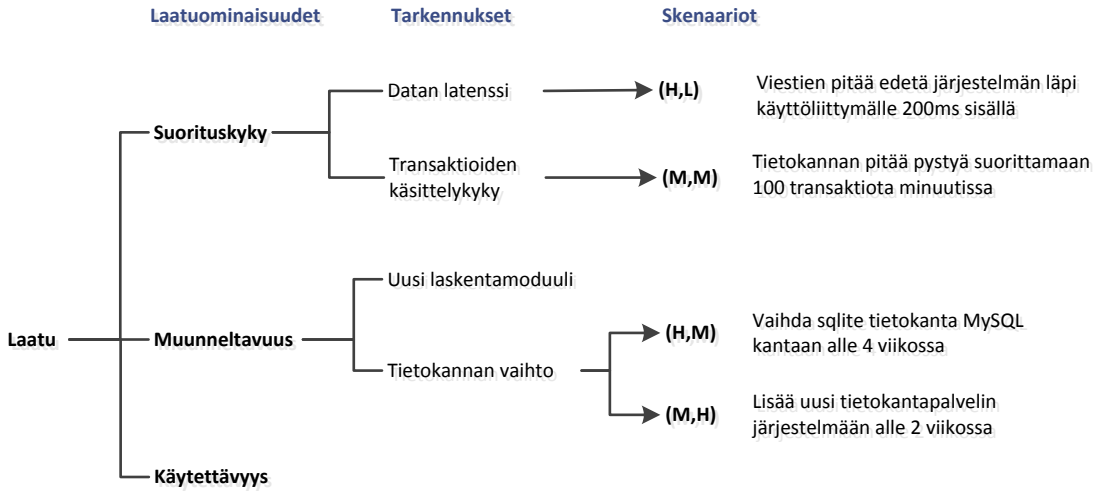
2.2.2 Laatuominaisuudet, laatupuu ja skenaariot

ATAM-pohjaisessa analyysissä arkkitehtuuria lähetään tutkimaan siltä vaadittuja laadullisia ominaisuuksia vastaan. Analyysin tekeminen vaatii, että yksittäiset laatuominaisuudet pitää pystyä tunnistamaan ja erottamaan toisistaan. ISO-standardi (International Organization for Standardization) määrittelee kahdeksan laatuominaisuutta, joista kukin koostuu pienemmistä osista, laatualiominaisuuksista. Kuva 2 esittää standardin määrittelemät laatuominaisuudet, joista *luotettavuus* on jaettu edelleen aliominaisuuksiinsa. ATAM-menetelmässä ei kuitenkaan tarvitse noudattaa ISO-standardin määrittelemiä laatuominaisuuksia orjallisesti, vaan niistä voi käyttää synonyymejä sovellettavan alan käsitteistöstä tai yhdistelmiä tilanteen salliessa.



Kuva 2 Ote ISO/IEC FCD 25010 laatukehikosta [3]

Analyysin yksi tärkeimmistä työkaluista on laatupuu (kuva 3). Laatupuu on nelitasoinen puurakenne, jonka juurena on *Laatu*. Laatupuun juureen yhdistyvä toinen taso kokoaa vaaditut laatuominaisuudet, joista kukin käsitellään erikseen. Laatuominaisuuksina on suositeltavaa käyttää ISO-standardin laatukehikon (kuva 2) määrittelemiä laatuominaisuuksia, mutta tarpeen mukaan standardin laatukehikon kategorioista voi joustaa. Laatuominaisuudet tarkennetaan vaatimuskategorioilla, jotta selviää miksi kyseistä laatuominaisuutta järjestelmältä vaaditaan. Tarkennuksista laaditaan erilaisia skenaarioita, joita käyttäen järjestelmä voidaan testata ja joissa laatuvaatimukset realisoituvat. Skenaariot arvioidaan karkeasti asteikolla: high (H), medium (M), low (L). Kuva 3 esittää pienen osan täytetystä laatupuusta, jossa skenaariot ovat viimeisen tason lehtinä. Kuvan skenaarioon on annettu kaksi arvoa: Ensimmäinen arvo kertoo kuinka tärkeä skenaario on ja toinen kuinka vaikea se on implementoida tai muuten toteuttaa.



Kuva 3 Osittain täytetty laatupuu

Laatupuun skenaariot käsitellään yksitellen ja ne ovat tärkeimmät työkalut arkkitehtuurin analysoinnissa. Skenaario kuvataan yksinkertaisimmassa tapauksessa rakenteella *ympäristö, ärsyke* ja *odotettu vaste*. Skenaarioon listataan kaikki arkkitehtuuriratkaisut, jotka liittyvät kyseiseen skenaarioon. Kukin arkkitehtuuriratkaisu analysoidaan yksitellen ja niiden vaikutus skenaariota vastaavaan laatuominaisuuteen

käydään tiimin kesken läpi. Kuva 4 esittelee esimerkkiskenaarion tietokannan suorituskyvystä. Skenaarion loppuun merkitään selkokielisesti perustelut suunnitteluratkaisuihin tehdyistä merkinnöistä ja siihen voi mainita myös muut löydökset. Loppuun voi lisätä myös skenaarioon liittyvät arkkitehtuurikuvat, jotka selventävät skenaariota.

Skenaario #S2	Tietokannan tulee pystyä suorittamaan 100 transaktiota minuutissa			
Ominaisuus	Suorituskyky			
Ympäristö	Normaali toiminta			
Ärsyke	Tietokanta saa 100 transaktiota ensimmäisen 10 sekunnin aikana			
Vaste	Tietokanta suorittaa kaikki transaktiot 60 sekunnin sisällä			
Suunnitteluratkaisu	**Herkkyyskohta**	**Tasapainokohta**	**Riski**	**Turvallinen ratkaisu**
Tietokantoja on yksi	S2	T1	R1	
Erillinen tietokantapalvelin	S3			N1
...				
Syyt	* Tietokannan koon kasvaessa suureksi transaktioiden suoritus hidastuu ja koska pyynnöt joudutaan sarjallistamaan, jolloin kannan koolla 100 transaktion suoritus kestää yli 60 sekuntia.			

Kuva 4 Esimerkkiskenaario tietokannan suorituskyvystä

Kuhunkin suunnitteluratkaisuun kirjataan merkintä jos niissä nähdään *herkkyyskohtia*, *tasapainokohtia* tai *riskejä*. Toisaalta, mikäli suunnitteluratkaisu nähdään *turvallisena ratkaisuna*, lisätään jälleen merkintä. Jokainen herkkyyskohta, tasapainokohta, riski ja turvallinen ratkaisu merkitään yksilöllisellä kirjain-numeroyhdistelmällä, jolla löydös kirjataan skenaarion lisäksi erilliseen katalogiin. Esimerkkiskenaarion tapauksessa (kuva 4) herkkyyskohdat on merkitty S-kirjaimella (sensitivity), tasapainokohdat T-kirjaimella (tradeoff), riskit R-kirjaimella (risk) ja turvalliset ratkaisut N-kirjaimella (non-risk). Nämä selvitetään tarkemmin seuraavassa aliluvussa. Mikäli useamman skenaarion piirin on tunnistettu sama suunnitteluratkaisu ja näihin skenaarioihin vaikuttaa esimerkiksi sama riski, kyseinen riski kirjataan kuhunkin skenaarioon samalla yksilöllisellä tunnisteella. [1]

2.2.3 Herkkyyskohdat, tasapainokohdat ja riskit

ATAM-analyysin edetessä arkkitehtuurin tarkempaan tarkasteluun suunnitteluratkaisuista tunnistetaan niin sanottuja *herkkyyskohtia*, *tasapainokohtia* ja *riskejä*. Nämä ovat arkkitehtuurin kohtia, jotka vaikuttavat suoraan järjestelmän laatuominaisuuksiin. Arkkitehti saattaa olla tietoinen näistä arkkitehtuurin ominaisuuksista jo suunnitteluvaiheessa, mutta niistä ei ole välttämättä mainintaa arkkitehtuurin dokumentaatiossa [1]. Herkkyys- ja tasapainokohdat sekä riskit ja turvalliset ratkaisut ovat tärkeitä ATAM-menetelmästä saatavia tuloksia.

Arkkitehtuurilliset *herkkyyskohdat* (engl. sensitivity point) ovat suunnitteluratkaisuja, jotka vaikuttavat yhteen laatuominaisuuteen. Herkkyyskohta on arkkitehtuuriratkaisu, joka on tärkeä jonkin laatuvaatimuksen saavuttamisen kannalta. Esimerkkinä palvelimien määrä vaikuttaa web-palvelun suorituskykyyn ja siten palvelimien

lukumäärä on herkkyyskohta. Toisaalta salatun yhteyden turvallisuus on ”herkkä” kryptauksessa käytettyyn bittimäärään. Mikäli herkkyyskohtaan vaikutetaan – tässä tapauksessa palvelimien määrää muutetaan tai salauksen bittimäärää muutetaan – muutos vaikuttaa suoraan herkkyyskohtaan liittyvään laatuominaisuuteen. [1][5]

Tasapainokohta (engl. tradeoff point) on kuin herkkyyskohta, mutta vaikutettavia laatuominaisuuksia on kaksi tai useampi. Laatuominaisuuksiin kohdistuva vaikutus on ominaisuuksien kesken yleensä vastakkaissuuntainen, mistä tulee nimitys ”tasapainokohta”. Usean palvelimen käyttö web-palvelussa on hyvä esimerkki tasapainokohdasta: palvelimien lisääminen parantaa suorituskykyä ja saatavuutta, mutta joissain useamman palvelimen ratkaisuissa tietoturvallisuus voi heikentyä. Tasapainokohdan negatiiviset laatuvaikutukset saattavat olla järjestelmän vaatimuksien kannalta hyväksyttäviä, mutta niiden tiedostaminen on hyödyllistä mikäli tasapainokohtaan tehdään muutoksia jossain vaiheessa järjestelmän elinkaarta. [5]

Arkkitehtuurista löydetyt *riskit* (engl. risk) ovat arkkitehtuuripäätöksiä, jotka potentiaalisesti estävät halutun laatuominaisuuden saavuttamisen. Esimerkiksi tietokantaa toteutettaessa yhden ohjelmaan integroidun SQLite-tietokannan käyttäminen voidaan nähdä riskinä muunneltavuuden kannalta, mikäli ohjelman rinnalle halutaan lisätä samaa tietokantaa käyttävä selaintuki. Koska SQLite ei perustu tietokantapalvelimen käyttöön, useamman samanaikaisen käyttäjän lisääminen vaatii muutoksia arkkitehtuuriin, mikä voi koitua hyvin ongelmalliseksi. Toisaalta analyysissä voidaan nähdä jokin suunnitteluratkaisu *turvallisena ratkaisuna* (engl. non-risk), eli ratkaisuna joka edesauttaa jonkin laatuominaisuuden toteuttamista [1]. MySQL-tietokantapalvelimen käyttö saatettaisiin nähdä tässä esimerkkitilanteessa turvallisena ratkaisuna.

Riskit ovat hyvin tärkeitä arkkitehtuurista saatavia tietoja, joiden löytyminen varhaisessa vaiheessa projektia saattaa johtaa huomattaviin säästöihin projektin kokonaiskustannuksissa. Lisäksi herkkyys- ja tasapainokohtien tunnistaminen on kehityksen kannalta hyödyllistä, sillä implementoinnin tai ylläpidon aikana saattaa tulla tarve tehdä muutoksia näihin järjestelmän kohtiin. Tieto siitä, miten eri arkkitehtuuriratkaisut vaikuttavat eri laatuominaisuuksiin, ohjaa tekemään turvallisempia muutoksia arkkitehtuuriin ja sen eri osiin.

2.2.4 Analyysin toimenpiteet

Analyysi jakautuu yhdeksään selkeästi toisistaan eroteltavaan toimenpiteeseen, jotka suoritetaan järjestyksessä alusta loppuun. Järjestykseen voi tehdä niissä tapauksissa poikkeuksia, kun halutaan siirtyä prosessissa taaksepäin selventämään tai täydentämään jotain aikaisempaa työvaihetta. Rakenne muodostuu karkeasti ilmaistuna toistuvista *ratkaisujen esittely ja analysointi* –työpareista.

Toimenpide 1: ATAM menetelmän esittely. Ensimmäisessä vaiheessa evaluoinnin johtaja esittelee ATAM-mentelmän perusteet ja prosessin kulun arviointiin osallistujille. Perinteisessä esittelyssä käydään lyhyesti läpi ATAM-menetelmän hyödyt ja piirteet,

menetelmän suorituksen vaiheet sekä analyysin lopputuotteet. Esittelyssä käydään vaiheita läpi usein esimerkkien avulla. Esittelyssä ei käydä menetelmää kuitenkaan perusteellisesti läpi, sillä riittää, että prosessin vetovastuussa oleva evaluointitiimi tuntee prosessin tarkemman kulun. Ensimmäisen työvaiheen kokonaiskesto on noin tunnin mittainen. [2]

Toimenpide 2: Liiketoimintanäkökulmien esittäminen. Jotta analyysin tuloksiin voisi luottaa, tulee sekä evaluointitiimin että muiden osakkaiden ymmärtää arkkitehtuurin konteksti ja siihen vaikuttavat tärkeimmät liiketoimintanäkökulmat. Liiketoimintanäkökulmat valottavat syitä arkkitehtuurillisiin suunnitteluratkaisuihin ja niihin asetettuja rajoitteita. Tämän esityksen pitää joku projektin päättäjistä, useimmiten joko projektipäällikkö, tai asiakas jolle järjestelmä toimitetaan. Esityksessä käydään läpi mm:

- järjestelmän tärkeimmät toiminnot ja konteksti
- liiketoiminnalliset tavoitteet ja rajoitteet
- arkkitehtuurillisesti tärkeimmät vaatimukset
- liiketoiminnan kannalta tärkeät laatuominaisuusvaatimukset
- tekniset rajoitteet. [1]

Toimenpide 3: Arkkitehtuurin esittely. Arkkitehtuurin esittely on lyhyt, noin tunnin mittainen esitys, jonka pitää järjestelmän pääarkkitehti. Evaluointitiimille on toimitettu etukäteen kaikki arkkitehtuurin dokumentaatio ja tässä esitelmässä pyritään käymään lyhyesti läpi vain arkkitehtuurin pääkohdat. Esitelmän sisällön tärkeimmät asiat ovat arkkitehtuurin suunnitteluratkaisujen esittely, sekä järjestelmälle asetetut vaatimukset ja rajoitteet. Arkkitehtuuri tulee esitellä niillä arkkitehtuurikuvaustavoilla, jotka olivat tärkeimmät arkkitehtuuria suunnitellessa sekä niillä kuvaustavoilla, jotka selvittävät, miten tärkeimmät laatuominaisuudet on saavutettu. [1]

Toimenpide 4: Arkkitehtuuriratkaisujen tunnistaminen. Arkkitehtuurin suunnittelussa käytetyt suunnitteluratkaisut on valittu sillä periaatteella, että järjestelmälle asetetut laatuvaatimukset voitaisiin täyttää. Kullakin lähestymistavalla voi olla sekä tavoiteltuja laatuominaisuuksia parantavia piirteitä että toisia, ominaisuuksia heikentäviä piirteitä. Esimerkiksi kerrosarkkitehtuuri parantaa siirreltävyyttä, mutta se voi joissain tapauksissa heikentää ylläpidettävyyttä. Tässä vaiheessa on tarkoitus koota lista käytetyistä arkkitehtonisista lähestymistavoista ATAM-prosessin myöhempien vaiheiden käyttöä varten. [1]

Toimenpide 5: Laatupuun ja skenaarioiden laatiminen. Laatupuun laadinta on selkeä tapa koostaa vaatimukset ja skenaariot yhteen selkeästi luettavaan rakenteeseen. Laatupuun lehdille ideoidaan skenaarioita, eikä esitettyjen ideoiden tarvitse olla täysin loppuun asti ajateltuja. Laaditut skenaarioideat jalostetaan sellaiseen muotoon, että niitä voi käyttää myöhemmissä työvaiheissa ja asetetaan puun lehdiksi. Skenaarioiden tulee

olla riittävän tarkkoja, jotta niiden perusteella voidaan testata arkkitehtuuria. Skenaarioita ideoidessa voi tulla julki täysin uusia, kirjaamattomia laatuvaatimuksia, jotka tulee lisätä laatupuun rakenteeseen. Toisaalta on mahdollista, että aikaisemmin tärkeäksi määriteltyyn laatuominaisuuteen ei ole saatu ideoitua skenaarioita lehdiksi ja se voidaan todeta vähemmän tärkeäksi laatuominaisuudeksi. Laatupuu esiteltiin tarkemmin aliluvussa 2.2.2. [1]

Kun skenaariot on laadittu ja asetettu puun lehdiksi, ne priorisoidaan asteikolla: high (H), medium (M), low (L). Arvio annetaan kahdesta asiasta: kuinka tärkeä skenaario on ja kuinka vaikea ominaisuus tai kokonaisuus on toteuttaa. Vaihtoehtoisiakin priorisointitapoja, kuten numeroita voi käyttää, mutta oleellisinta on saada karkea prioriteettijako skenaarioiden välille. Skenaarioita on laadittu todennäköisesti huomattavasti enemmän kuin evaluoinnissa on aikaa käsitellä, joten priorisoinnilla saadaan valittua tärkeimmät skenaariot. Ensisijaisesti valitaan (H,H) skenaariot, joiden jälkeen käsitellään (H,M) ja (M,H) skenaariot. Muihin skenaarioihin mennään vain jos aika riittää. [1][2]

Toimenpide 6: Arkkitehtuuriratkaisujen analysointi. Suunnitteluratkaisujen analysointi on työvaihe, jossa vaaditaan syvällistä arkkitehtuurituntemusta. Analysointivaiheessa toimenpide 5:n tärkeimmät skenaariot puretaan osiin ja analysoidaan. Skenaariot kuvataan, niihin merkitään arkkitehtuuriratkaisut, jotka vaikuttavat skenaarioon ja kukin suunnitteluratkaisu käydään yksitellen läpi. Kustakin skenaariosta löydetyt *herkkyyskohdat*, *tasapainokohdat* ja *riskit* kirjataan ylös siten, että analyysin lopuksi on saatavilla tarkka lista löydöksistä ja niiden perusteluista. Toimenpide 6 päättää ATAM-analyysin ensimmäisen vaiheen. Tarkempi kuvaus skenaarioista ja niiden analysoinnista on esitelty aliluvussa 2.2.2.

Toimenpide 7: Skenaarioaivoriihi. Toimenpide 7 on hyvin samanlainen kuin toimenpide 5 sillä erotuksella, että tärkeimmät ideoinnin osallistujat ovat tässä vaiheessa evaluointiin liittyneet muut sidosryhmät. Tämän aivoriihen tarkoitus on tuoda loppukäyttäjän, asiakkaan ja muiden toimijoiden näkökulmat mukaan. Näin arkkitehtuuri saadaan testattua niitä skenaarioita vasten jotka todennäköisimmin toteutuvat. Skenaarioista valitaan työvaihe 5:n tavoin tärkeimmät, mutta valintatapa on erilainen. Kukin osakas saa tietyn määrän ”pisteitä” jaettavaksi skenaarioiden kesken ja kukin jakaa pisteensä sen mukaan, mitkä skenaariot hän haluaa käsiteltävän. Kun kaikki osakkaat ovat jakaneet pisteensä, skenaariot saadaan tärkeysjärjestykseen ja näistä valitaan tärkeimmät. Evaluointitiimi päättää, mihin kohtaan priorisoitua listaa vedetään analyysiin valittavien skenaarioiden raja. [1]

Skenaarioiden ideoinnissa osakkaita kehotetaan yleensä laatimaan mm. *käyttötapausskenaarioita*, *kasvuskenaarioita* ja *tutkivia skenaarioita*. *Käyttötapaus-skenaariossa* kartoitetaan sitä, miten osakkaat olettavat järjestelmää käytettävän. *Kasvuskenaarioissa* ideoidaan tapoja miten järjestelmää oletetaan kasvatettavan ja muunnettavan. Kasvuskenaariot voivat kattaa muutoksia esimerkiksi suorituskykyyn tai

saatavuuteen. *Tutkivat skenaariot* etsivät arkkitehtuurin rajoja tavoilla, joissa esimerkiksi kasvuskenaarion vaatimukset vaaditaan kertaluokkaa suurempina. Tutkivissa skenaaroissa arkkitehtuurista löydetään yleensä lisää herkkyys ja tasapainokohtia niistä kohdista, mistä skenaario rasittaa arkkitehtuuria eniten. [1]

Toimenpide 8: Arkkitehtuuriratkaisujen uusinta-analysointi. Tämä työvaihe on lähes identtinen työvaihe 6:n kanssa. Mikäli aikaisemmissa vaiheissa on onnistuttu hyvin, tässä vaiheessa ei pitäisi tulla enää paljon tai yhtään uusia tasapainokohtia tai riskejä. Tästä syystä toimenpiteitä 7 ja 8 kutsutaan myös eräänlaisiksi testausvaiheeksi. Analysoitavia skenaarioita käsitellään 5 – 10 jäljellä olevasta ajasta riippuen. [1]

Toimenpide 9: Tulosten julkistaminen. Analysointipäivän viimeisenä toimenpiteenä on esitellä kooste analyysin tuloksista. Tärkeimmät tulokset ovat:

- lista arkkitehtuurin suunnitteluratkaisuista
- skenaariot ja niiden priorisoinnit
- laatupuu
- lista riskeistä ja turvallisista ratkaisuista
- lista herkkyys- ja tasapainokohdista.

Edellämainitun listan lisäksi evaluointitiimi koostaa löydetyt riskiteemat, jotka on havaittavissa useista toisiinsa liittyvistä riskeistä. Lisäksi esitetään muut arkkitehtuuriin tai projektiin liittyvät yleiset huolenaiheet, jotka ovat tulleet evaluoinnin aikana esille. Näitä on esimerkiksi puutteellinen dokumentaatio tai asiakkaiden heikko luottamus toimittajan suorituskykyä kohtaan. Tulokset esitetään sekä suullisesti esitelmän muodossa että paperimuodossa, jotta tulokset olisivat arkistoitavissa. [1]

2.2.5 Analyysin jako neljään vaiheeseen

Eri lähdemateriaaleissa esitellään useita erilaista vaihejakoa, joista toisessa jaetaan neljään vaiheeseen ja toisessa kahteen tai vain yhteen vaiheeseen. Metodin kevennetyissä variaatioissa on usein jätetty osa vaiheista pois ja keskitytty analysointivaiheeseen/-vaiheisiin. Oppikirjanmukaisessa ATAM-menetelmässä on neljä vaihetta: vaiheet 0 – 3.

Vaihe 0 on valmisteluvaihe, jossa evaluointitiimi saa arkkitehtuurimateriaalit tutkittavaksi etukäteen. Evaluoinnin johtajan tehtävä on päättää saatujen materiaalien perusteella, onko saatu informaatio riittävä evaluoinnin suorittamiseen. Mikäli materiaalia on liian vähän, johtaja voi tehdä päätöksen, että arkkitehtuuria ei analysoida, ellei dokumentaatiota paranneta riittävälle tasolle. Valmisteluvaiheessa ATAM-analyysin tilannut asiakas perehdytetään siihen, miten evaluointi viedään läpi ja mitä tuloksia siitä saadaan irti.

Vaihe 1 ja 2 ovat itse analyysivaiheet, joiden toimenpiteet on esitetty aliluvussa 2.2.4. Vaihe 1 on ensimmäinen evaluointikierros, johon kokoontuu pienempi määrä ihmisiä (3-5 henkilöä) kuin toiseen vaiheeseen. Ensimmäisen vaiheen tarkoituksena on

tehdä päätös kannattaako toiseen vaiheeseen siirtyä ja tarvitaanko lisää dokumentaatiota, ja jos tarvitaan niin minkälaista. Lisäksi tässä vaiheessa kartoitetaan, mitä sidosryhmiä toiseen vaiheeseen halutaan osallistuvan. Myös vaiheen 2 evaluointitiimin kokoonpano päätetään tässä riippuen arkkitehtuurin vaatimuksista ja siitä, ketkä evaluoijat tuntevat vaatimuksiin liittyvän alan parhaiten. Esimerkiksi turvallisuuskriittisen järjestelmän evaluointiin valitaan vähintään yksi tietoturvallisuuden hyvin tunteva asiantuntija.

Vaihe 2 on täyden evaluoinnin kierros, joka suoritetaan kaksi- tai kolmepäiväisenä. Toiseen vaiheeseen osallistuu evaluointitiimin ja projektin päättäjien lisäksi tärkeimmät sidosryhmät, joiden läsnäolo on nähty tarpeelliseksi. Eri sidosryhmiä on monia ja valitut sidosryhmät voisivat olla esimerkiksi ylläpitäjä, markkinoija, tuotepäällikkö, testaaja ja loppukäyttäjä. [1] Sidosryhmien merkitys korostuu aivoriihessä ja analysoinnissa (vrt. aliuku 2.2.4), eikä kaikkien osakkaiden ole tarpeellista osallistua ensimmäisen päivän toimenpiteisiin. Pienemmän mittakaavan ATAM-menetelmässä suoritetaan vain vaihe 2:n toimenpiteet, ja ensimmäisen vaiheen merkitys vähenee, mikäli evaluaatiotiimin tai sidosryhmien kokoonpanoon ei pystytä vaikuttamaan. Taulukko 2 esittelee toimenpidejaon vaiheille 1 ja 2.

Taulukko 2 Vaiheiden 1 ja 2 toimenpiteet

	1. Päivä	Vaihe 1 & 2 toimenpiteet
1	ATAM esittely	
2	Liiketoimintanäkökulmien esittely	
3	Arkkitehtuurin esittely	
4	Suunnitteluratkaisujen tunnistaminen	
5	Laatupuun ja skenaarioiden laadinta	
6	Suunnitteluratkaisujen analysointi	
	2. Päivä	Vaihe 2 toimenpiteet
1	ATAM esittely	
6	Skenaarioiden nopea läpikäynti	
7	Skenaarioaivoriihi	
8	Uusinta-analysointi	
9	Tulosten julkistaminen	

Vaihe 3 on päätösvaihe, jossa koostetaan tulokset. Päätösvaiheessa asiakkaalle toimitetaan ATAM:n löydöksistä kirjallinen raportti, joka koostuu siitä, mitä on tehty, mitä on löydety ja mitä johtopäätöksiä on tehty. Evaluoiva taho tekee töitä myös itselleen päätösvaiheessa. Päätösvaiheessa kerätään tietoja siitä, kuinka prosessin kulku on mennyt ja kuinka sitä voi jatkossa parantaa. Lisäksi tehty analyysi arkistoidaan tapaustutkimuksena myöhempää koulutuskäyttöä varten. Arkistoituun versioon on kuitenkin tehtävä tarvittavat muutokset, jotta mahdolliset yhtiösalaisuudet ei käy ilmi [1].

ATAM jakautuu edellä mainittuihin neljään selkeään vaiheeseen, jotka suoritetaan järjestyksessä, mutta vaiheet ovat huomattavan eri mittaisia. Vaihe 0, valmisteluvaihe,

voi kestää useita viikkoja ja siihen lasketaan myös se valmistelutyö, mitä on tehty ilman työstä allekirjoitettuja sopimuksia. Vaiheet 1 ja 2 ovat lyhyitä, kokonaisuudessaan kolmesta neljään päivään, mutta välissä voi olla parin viikon tauko. Tauon aikana koostetaan ensimmäisen evaluoinnin tuloksia ja valmistaudutaan toista evaluointia varten, jotta siitä saataisiin mahdollisimman paljon irti. Viimein evaluoinnin päätyttyä alkaa päätösvaihe, joka kestää noin viikon. Tänä aikana on tarkoitus tuottaa tulokset ja päättää ATAM-menetelmän mukainen analyysi.

2.3 ATAM-menetelmän tulokset ja sivutuotteet

Edellisissä luvuissa on tullut ilmi yksittäisiä lopputuotteita ja sivutuotteita, joita ATAM-evaluointi tuottaa. Prosessin ensimmäisenä sivutuotteena varmistetaan, että arkkitehtuurista on tehty riittävän laadukas dokumentaatio. Puutteellinen dokumentaatio on jo projektinhallinnallisesti riskitekijä, mihin puututaan ATAM-menetelmässä jo heti alkuvaiheessa. Prosessi pakottaa arkkitehdin tuottamaan arkkitehtuurista sekä ymmärrettävän esityksen että laadukkaan dokumentaation, jossa järjestelmä ja siinä käytetyt suunnitteluratkaisut tulevat selkeästi ilmi. Mikäli arkkitehtuurista on tehty liian vähän dokumentaatiota, tai dokumentaation tarkkuus ei ole riittävä, dokumentaatio ei ole tarpeeksi hyvä ATAM-analyysiä varten. Mikäli dokumentaatio nähdään heikoksi, ATAM-menetelmän alkuvaiheessa evaluoijien antama palaute ohjaa arkkitehtiä korjaamaan dokumentaation puutteet.

Prosessin keskivaiheilla työstetään laatupuu, priorisoidut skenaariot ja suunnitteluratkaisut. Järjestelmältä vaadittavat tärkeimmät laadulliset ominaisuudet on listattu laatupuuhun heti ensimmäiselle alitasolle. Kukin laatuominaisuus on jäsennelty tarkennuksineen, jotka jakautuvat edelleen omiin skenaarioihinsa. Laatupuu itsessään antaa kokonaiskuvan siitä, mitä laatuominaisuuksia järjestelmältä odotetaan ja mihin vaatimuksiin sen pitää vastata. Laatupuun lehtinä olevista skenaarioista saadaan priorisoitu lista, josta käy ilmi mitkä skenaariot ovat tärkeimmät ja hankalimmat toteuttaa, ja mitkä skenaariot jäävät priorisoinnissa vähemmän tärkeiksi.

Tulosten tärkeimmästä päästä ATAM antaa skenaarioiden analysoinnissa listan herkkyyskohdista, tasapainokohdista, riskeistä ja riskiteemoista. Siinä, missä riskit kertovat järjestelmästä löytyneet yksittäiset riskit, riskiteemat niputtavat useammat riskit yhteen. Riskiteemoissa voi käydä ilmi esimerkiksi että järjestelmässä nojaudutaan liikaa kaupallisiin tuotteisiin. Toisaalta riskiteemoissa voidaan huomauttaa, että missään dokumentissa ei käydä läpi yleiskuvaa ja siten tarkkakin dokumentaatio on vaikeasti ymmärrettävä [1]. Skenaarioissa käsitellyt arkkitehtuurin suunnitteluratkaisut sekä muut skenaarion analyysin tulokset listataan omaan dokumenttiinsa.

Muihin tärkeisiin ulostuloihin lukeutuvat selkeytyneet liiketoimintanäkökulmat, jotka osaltaan vaikuttavat laatupuuhun ja skenaarioihin. On mahdollista, että osa kehittäjistä kuulee liiketoimintanäkökulmista ensimmäistä kertaa vasta ATAM-menetelmän aikana [1]. Liiketoimintanäkökulmat antavat perustelun osalle järjestelmälle annetuista vaatimuksista, mutta toisaalta niiden myötä on pääteltävissä toistaiseksi kirjaamattomia

vaatimuksia. Liiketoimintanäkökulmat eivät välttämättä tule kirjalliseen muotoon koostettuna dokumenttina, mutta ne ovat silti hyvin tärkeä prosessin sivutuote.

2.4 Analysoinnin edut ja kustannukset

Mikäli analyysin järjestää arkkitehtuurista vastaava organisaatio, analysoinnin kustannukset ovat mitattavissa lähinnä osallistujien käyttämässä ajassa. Analyysi, johon osallistuu myös asiakas, ei välttämättä tule maksamaan pelkästään arkkitehtuurista vastaavalle organisaatiolle, sillä asiakkaat voivat olla halukkaita osallistumaan omaan laskuunsa [1]. Käytetty aika riippuu paljolti osallistujien määrästä, mutta analyysin aloittamista ja suorittamista saattaa hidastaa myös organisaation heikko arkkitehtuurillinen maturiteetti [6].

Suora rahallinen hyöty on mitattavissa arkkitehtuuriin käytettävistä resursseista suhteessa arvioituihin kustannuksiin, jos projekti olisi tehty ilman analyysin pohjalta tehtyjä muutoksia. Vuonna 1997 AT&T raportoi kahdeksan vuoden ajalla tehtyjen arkkitehtuurien evaluointien pohjalta, että arkkitehtuurianalyysi on pienentänyt projektin kustannuksia keskimäärin kymmenen prosenttia. Täyden mittakaavan ATAM-evaluoinnin viedessä 70 miestyöpäivää, evaluointi alkaa vähentämään projektin kustannuksia projektin pituuden ylittäessä 700 miestyöpäivää. Toisaalta on myös olemassa esimerkkejä pahasti epäonnistuneista projekteista, joissa ei ole suoritettu minkäänlaista evaluointia ja joihin on palanut mittaamattomasti rahaa [6]. Mikäli näissä tapauksissa analyysi olisi tehty varhaisessa vaiheessa, olisi ollut mahdollista joko korjata arkkitehtuuria tai hylätä projekti, mikäli vaatimuksien täyttäminen olisi nähty mahdottomaksi.

Arkkitehtuurin evaluoinnissa on useita eri tekijöitä jotka hyödyttävät sekä projektia että organisaatiota. Ensimmäisenä näkyvät hyödyt ovat dokumentaation paraneminen sekä järjestelmän parempi ymmärtäminen. Arkkitehtuurin ongelmakohtien löytymisen aikana myös vaatimukset selkeytyvät ja priorisoituvat [6]. On mahdollista, että vaatimuksissa on selvästi havaittavia ristiriitoja, jotka saattavat aiheuttaa projektin myöhäisemmässä vaiheessa ongelmia. Ristiriitaisten vaatimusten priorisointi ja muuttaminen on suunnitteluvaiheessa huomattavasti edullisempaa kuin implementointivaiheessa [2]. Lisäksi perusteellisesti tehdyllä evaluoinnilla on mahdollista valaa kaikkiin osakkaisiin uskoa arkkitehtuurin suhteen ja asiakkaan uskoa toimittajaa kohtaan [6]. Tämän uskon valamisen tärkeys kasvaa etenkin silloin, kun kyseisen asiakkaan usko on vähentynyt toimittajan kykyyn toimittaa järjestelmä ennalta määritetyssä aikataulussa ja budjetissa.

Organisaatiot, joissa arkkitehtuureja analysoidaan osana kehitysprosessia, ovat havainneet selkeää parannusta arkkitehtuuriensa yleisessä laadussa. Tämä johtuu siitä, että arkkitehti osaa arkkitehtuuria suunnitellessa ottaa tulevan evaluoinnin huomioon ja panostaa sitä varten. Arkkitehti osaa odottaa, minkälaisia kysymyksiä evaluoinnissa esitetään ja mitä materiaaleja siellä vaaditaan. Pidemmän ajan kuluessa on havaittavissa,

että organisaation kyky suunnitella parempia arkkitehtuureja kehittyy ja organisaatio on kehittänyt yleisiä toimintatapoja tuottaa parempia arkkitehtuureja. [2][6]

2.5 Muut evaluointimenetelmät

ATAM ei ole ainoa metodi, jolla voi analysoida ohjelmistoarkkitehtuureja. ATAM-menetelmä on SEI:n kehittämä tekniikka, joka on kehitetty heidän SAAM-menetelmän (Software architecture analysis method) pohjalta. SAAM-pohjainen analyysi on ATAM:ia suppeampi ja keskittyy lähinnä muunneltavuuden ja toiminnallisuuden analysointiin [4]. SEI on kehittänyt ATAM-menetelmän tueksi QAW-menetelmän (Quality Attribute Workshop), joka keskittyy laatuominaisuuksien kartoittamiseen ja pitkälle jalostettujen skenaarioiden tekemiseen. QAW-menetelmän skenaariot ovat suoraan käytettävissä ATAM-analyysin materiaaleina. [7]

ATAM:n selvittäessä minkälaisia riskejä arkkitehtuurissa on ja mitä laatuominaisuuksia se täyttää, se ei kuitenkaan puutu kustannusseikkoihin. CBAM (Cost Benefit Analysis Method, myös SEI:n kehittämä tekniikka) pureutuu juuri näihin seikkoihin. CBAM-menetelmässä analysoidaan arkkitehtuuriratkaisujen kustannuksia, hyötyjä, epävarmuustekijöitä ja implementoinnin ajanhallinnallisia seikkoja [8]. CBAM-menetelmää voi käyttää luonnollisena jatkeena ATAM-menetelmän jälkeen, tai sen voi jopa integroida ATAM-menetelmän kanssa. [9]

ATAM, SAAM ja CBAM ovat kukin vain esimerkkejä useista eri skenaariopohjaisista menetelmistä. Skenaariopohjaisuuden lisäksi on olemassa muita tapoja evaluoida ja nämä muut ovat karkeasti jaettuna kyselykaavakepohjaisia, tarkistuslistapohjaisia, metriikkapohjaisia tai protyyppipohjaisia metodeja. Nämä menetelmät on karkeasti jaettavissa kahteen kategoriaan: kyselypohjaiset tekniikat ja mittauspohjaiset tekniikat. Mittauspohjaiset tekniikat ovat suppeita ja vastaavat lähinnä suorituskyky- ja muunneltavuuskysymyksiin. Kyselypohjaisista tekniikoista kysely-kaavake- ja tarkistuslistametodit ovat nopeita tapoja evaluoida, mutta analyysi ei ole välttämättä syväluotaavaa ja listojen kysymykset voivat olla sisällöltään helposti liian yleisluontoisia. Kyselypohjaisista tekniikoista viimeinen, skenaariopohjainen tekniikka puolestaan mahdollistaa tarkemman analyysin, mutta se näkyy suoraan evaluoinnin kustannuksissa. [6]

Evaluointitapoja on siis monia erilaisia. ATAM-menetelmän ollessa vain yksi monista se erottuu edukseen siinä, kuinka kattavasti sillä voi arkkitehtuurin analysoida. Mikäli käytetyltä menetelmältä halutaan vastauksia kysymyksiin mihin ATAM ei vastaa, tai halutaan edullisempi ja nopeammin suoritettava metodi, on olemassa useita muitakin metodeja, joista jokin voi vastata paremmin tarpeisiin. Evaluointi on kuitenkin prosessi, minkä tehokas suorittaminen vaatii kokemusta ja siten on viisasta harkita mitä menetelmää käyttää.

2.6 Yhteenveto

ATAM on prosessina hyvin selkeä ja työvaiheet on määritelty tarkkaan ja jopa aikataulustakin voi tehdä minuuttiaikataulun. Pelkkä kirjatietous ei kuitenkaan riitä menetelmän hyvään suorittamiseen, vaan prosessin menestyksekäs läpiajo vaatii evaluointitiimiltä vankkaa arkkitehtuurillista kokemusta. Laatuominaisuuksien, suunnitteluratkaisujen ja riskien tunnistaminen ovat menetelmän onnistumisen kannalta oleellista, ja kokemattomalla tiimillä jotain oleellista saattaa jäädä huomaamatta. Menetelmän toistuva suorittaminen on kuitenkin oiva tapa kartuttaa kokemattoman tiimin tai yksittäisten tiimiläisten kokemusta, jotta tulevat evaluoinnit olisivat hedelmällisempiä.

Vaikka täysimittaisen ATAM-menetelmän käyttö voikin olla kallista, sen suorittaminen ei vie kovin paljon kalenteriaikaa. Menetelmän tuomat säästöt ovat tuntuvia vain, jos projektin mittakaava on tarpeeksi suuri. Mikäli osallistujien määrää vähennetään ja suoritettuja vaiheita kevennetään, ATAM on sovellettavissa myös pienemmässä mittakaavassa, jolloin siitä tulee käyttökelpoinen työkalu myös pienempiin projekteihin. Kevennetyssä menetelmässä on kuitenkin mahdollisuus, että osa arkkitehtuurin riskeistä jää huomaamatta. Vaikka kaikkia riskejä ei saataisikaan kiinni, menetelmä maksaa nopeasti itsensä takaisin, jos arkkitehtuurista on löydettävissä vakavia riskejä.

3 ANALYSOITAVA OHJELMISTO

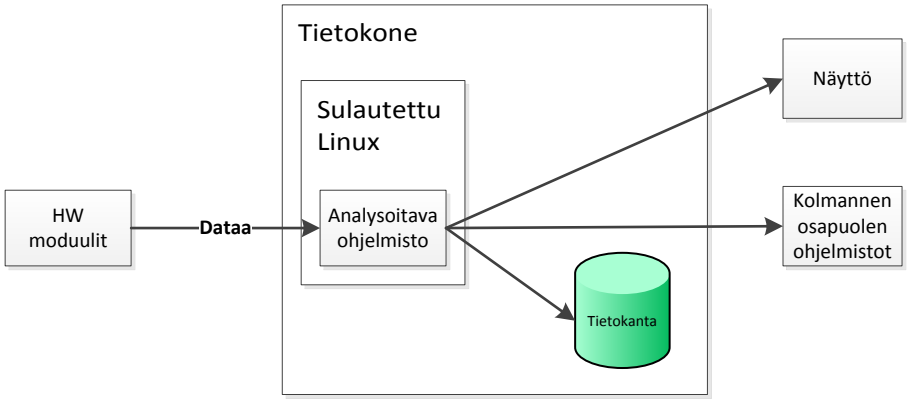
Tässä luvussa esitellään arkkitehtuuri, johon ATAM-menetelmää on sovellettu. Ohjelmisto ja sen käyttökohteet on kuvattu suppeasti, osittain, koska ohjelmiston koko on melko mittava, ja osittain, jotta ohjelmiston tarkempi käyttökohde pysyisi salassa. Arkkitehtuurin kuvaus on esitetty sillä tarkkuudella, että myöhemmin luvussa 4 esitetty menetelmän suoritus ja skenaariot olisivat helposti ymmärrettävissä. Ohjelmisto on jo valmis, joten tämän diplomityön puitteissa tehty analyysi on tehty valmiista tuotteesta. Analysoitavaan ohjelmistoon viitataan seuraavissa luvuissa vain nimellä ohjelmisto, tai monitorointiohjelma.

Tämä luku jakautuu rakenteellisesti useampaan eri osaan. Ensin aliluvussa 3.1 esitellään ohjelmiston tehtävät tekniseltä näkökulmalta. Todellisella käyttöympäristöllä ja lopullisilla käyttötapauksilla ei ole ohjelmiston toiminnan ymmärtämisen kannalta kovin paljon merkitystä. Ohjelmiston esittelyn jälkeen esitellään aliluvussa 3.2 vaatimukset ja ominaisuudet, joita ohjelmistolta odotetaan. Aliluku 3.3 kattaa teknisen kuvauksen koko arkkitehtuurista ja on siten aliluvun 4.4 skenaarioiden ymmärtämisen kannalta erittäin tärkeä. Lopuksi aliluvussa 3.4 käydään läpi tärkeimmän laatuominaisuuden, varioinnin, vaikutusta arkkitehtuuriin ja aliluvussa 3.5 niitä haasteita, joita tehdyt arkkitehtuurilliset lähestymistavat aiheuttavat.

3.1 Analysoitava ohjelmisto ja sen tehtävät

Analysoitava ohjelmisto on monitorointi- ja tallennusohjelmisto, minkä päätehtävä on lukea CAN-väylältä (engl. Controller Area Network bus) dataa, esittää sitä reaaliaikaisesti käyttäjälle ja tallentaa tietokantaan myöhempää tarkastelua varten. Ohjelmisto on kehitetty käyttäen Qt 4.x kehystä ja kohdealustana toimii sulautettu Linux-käyttöjärjestelmä. Käyttöjärjestelmä on peitetty käyttäjältä siten, että tietokoneen käynnistyttyä monitorointiohjelma on ainoa käyttäjälle näkyvä ohjelma, eikä käyttäjällä ole pääsyä konsoli- tai työpöytäohjelmiin.

Tietokoneen käynnistyttyä monitorointiohjelmisto aloittaa datankeruun automaattisesti. Vastaanotettu data jäsennetään yleiskäyttöiseen muotoon ja datan pohjalta suoritetaan laskentaa, minkä jälkeen data on käyttäjän ja tietokannan kannalta hyödyllisessä muodossa. Näiden lisäksi samaa jalostettua dataa voidaan välittää lähiverkkoon liitettyihin kolmannen osapuolen ohjelmistoihin niin kutsutun reaaliaikadatan rajapinnan (myöhemmin RT API, engl. Real-Time Application Programming Interface) kautta. Tämä kaikki tapahtuu riippumatta siitä, mitä käyttäjä tekee käyttöliittymällä, sillä tietokoneen käynnistyttyä käyttäjällä on oikeudet vain tarkastella dataa. Kuva 5 esittää järjestelmän yksinkertaistetun rakenteen.



Kuva 5 Yksinkertaistettu tietovuokaavio järjestelmästä

Data etenee järjestelmässä pääosin yksisuuntaisesti oheislaitteilta monitorointiohjelmistolle ja siitä eteenpäin. Tietokoneessa on integroitu laitteistotuki mm. CAN-väylille ja sarjaporteille. Lopullinen väyläkonfiguraatio voi vaihdella eri varianttien välillä. Monitorointiohjelmisto lähettää hyvin vähän dataa väylille ja sekin data on pääsääntöisesti tietojen pyytämistä. CAN-väylään kytketyt oheislaitteet lähettävät automaattisesti jatkuvasti dataa ilman, että sitä pitää erikseen pyytää, ja erillistä pyyntöä vaativien viestien määrä on murto-osa koko väyläliikenteestä. Monitorointiohjelmiston pääasiallinen tiedonlähde on CAN-väylä, joten seuraavissa arkkitehtuurikuvauksissa keskitytään eniten siihen, miten CAN-data etenee arkkitehtuurin läpi.

3.2 Arkkitehtuurisuunnittelua ohjaavat vaatimukset ja ominaisuudet

Monitorointiohjelmiston vaatimukset on asiakkaan puolesta erittäin tarkkaan määritelty käyttöliittymän kosmeettisia yksityiskohtia myöten. Suuri osa vaatimuksista määräytyi loppukäyttäjän vaatimien toiminnallisuuksien pohjalta, jotka muodostivat ohjelmiston ensisijaiset vaatimukset. Näiden lisäksi arkkitehtuurisuunnitelmissa oli otettava huomioon asiakkaan liiketoimintanäkökulmat, joista moni tähtäsivät tulevaisuuden tuotekehityskustannuksien minimoimiseen. Ensimmäisessä aliluvussa on käsitelty arkkitehtuurille asetettuja yleisiä vaatimuksia, sekä laatuominaisuuksia (aliluku 3.2.1) ja myöhemmin toisessa aliluvussa liiketoimintanäkökulmia (aliluku 3.2.2). Vaatimuksia tulee tämän aliluvun lisäksi ilmi myöhemmin arkkitehtuurin kuvauksessa (aliluku 3.3).

3.2.1 Yleiset vaatimukset ja laatuominaisuudet

Tärkein arkkitehtuurille asetettu laatuvaatimus on varioitavuus. Ohjelmistosta on räätälöity useita eri tuotteita, joissa on vain osittain yhtenevät toiminnallisuudet. Tästä johtuen toiminnallisuuksia pitää olla helppo lisätä ja poistaa tuotteesta. Osa ominaisuuksista saattaa olla kahden eri tuotteen välillä pääosin samanlaiset, mutta toiminnassa voi olla eroavaisuuksia. Tästä johtuen suurin osa moduulien toiminnallisuuksista pitää olla alusta lähtien konfiguroitavissa, jotta moduuli olisi

käytettävissä uudelleen useaan eri tuotteeseen. Varioitavuus on johtanut siihen, että arkkitehtuuri koostuu kolmenlaisista elementeistä: kaikille tuotekonfiguraatioille yhteisestä ydinkoodista, yksittäisiä ominaisuuksia toteuttavista moduuleista ja moduulien konfiguraatiotiedostoista.

Ohjelmistolle ei ole asetettu kovia reaaliaikavaatimuksia, mutta suorituskyvyn tulee olla hyvä. Arkkitehtuurin odotettu elinkaari on hyvin pitkä ja on odotettavissa, että kyseisen arkkitehtuurin pohjalta tehdään useita eri tuotekonfiguraatioita. Koska tulevien tuotekonfiguraatioiden vaatimuksia ei vielä tunneta, tulee varmistua, että arkkitehtuuri selviää kuormituksen kasvaessa. Osaltaan haasteita tuo se, että tietokantaan ja RT-rajapintaan halutaan päivitykset kaikista sisään tulevista viesteistä, mikä aiheuttaa kuormaa kaikille kerroksille. Päivitystahdille on kuitenkin määritelty yläraja, jolloin ohjelman ylikuormitus ei ole niin suuri riski. On mahdollista, että tulevissa tuotekonfiguraatioissa on kytkettynä enemmän dataa tuottavia oheislaitteita, tai tuotteeseen tarvitaan uusia raskaita ominaisuuksia, joten arkkitehtuurin tulee olla alusta lähtien mahdollisimman kevyt.

Ohjelmiston asennukseen liittyvä vaatimus on, että ohjelmisto tulee olla asennettavissa kohdetietokoneeseen siirrettävästä mediasta, kuten USB-muistitikulta (engl. Universal Serial Bus) ja asennustoimenpiteiden määrän tulee olla minimoitu. Asennuksen yhteydessä kohdetietokoneeseen tulee asentua karsittu Linux-käyttöjärjestelmä ja mahdollinen edeltävä asennus poistetaan. Mikäli tietokoneella on jo valmiina edeltävä versio asennettavasta ohjelmistosta, ohjelmiston tuottamat datat tulee säilyä omalla osiollaan asennuksen yli. Asennuspaketista tulee olla tehtävissä myös päivityspaketti, jonka pystyy asentamaan edellisen ohjelmaversion käyttöliittymän kautta turvautumatta esimerkiksi Linux-käyttöjärjestelmän komentorivikonsoliin.

3.2.2 Liiketoimintanäkökulmat

Tärkein syy uuden arkkitehtuurin uudistamiseen oli tulevaisuuden tuotekehityskustannuksien pienentäminen. Tuotteen edeltävä koodihaara oli kasvanut suuremmaksi ja monimutkaisemmaksi kuin silloinen arkkitehtuuri salli ja siten silloinen arkkitehtuuri oli tullut elinkaarensa päähän. Tuotteen jatkokehitys oli edeltävällä arkkitehtuurilla hidasta ja pienillä koodimuutoksilla saattoi olla arvaamattomia vaikutuksia koko ohjelmiston kannalta. Asiakkaan tulevaisuudennäkymät osoittivat, että tuotteen elinkaari tulee olemaan pitkä ja varianttien määrä tulee kasvamaan, joten arkkitehtuurin kokonaisvaltainen uusiminen paremmaksi tulisi pitkällä aikavälillä maksamaan itsensä takaisin.

Jotta uuden arkkitehtuurin tuotekehityskustannukset pysyisivät matalana, seuraavat laaatuominaisuudet nousivat hyvin tärkeään asemaan: muunneltavuus, varioitavuus ja konfiguroitavuus. Tuotteen tulee olla muunneltava ja konfiguroitava, jotta tuote pystyy toimimaan täydellä kapasiteetilla eri kohderaudoissa ja käsittelemään erilaisten oheislaitteiden viestejä. Lisäksi tuotteen tulee olla siten varioitava, että uutta tuotekonfiguraatiota tehtäessä kaikki edeltäviin tuotekonfiguraatioihin tehdyt moduulit ovat käytettävissä ilman koodin kopiointia tai moduulien uudelleenkoodaamista. Nämä

olivat laatuominaisuuksia, joihin edeltävä arkkitehtuuri ei pystynyt vastaamaan ja jotka olivat lähes ainoa syy uuden arkkitehtuurin toteuttamiseen.

Ohjelmistoprojekteissa on hyvin tavallista, että tuotteesta löytyy ohjelmistovirheitä. Tämän tuotteen kohdalla ohjelmointivirheisiin suhtaudutaan tavallista vakavammin ja jopa pienet kosmeettiset virheet tulee saada kitkettyä pois. Tämä lisää tuotteen testattavuuden tärkeyttä, jotta ohjelmointivirheet olisi helpommin löydettävissä. Kattavaa testausta voi tehdä sekä manuaalisesti että automatisoidusti. Molemmat keinot vaativat kattavat rajapinnat testaussyötteiden antamiseen ja vasteiden lukemiseen. Vaikka virheiden etsiminen onkin testaajan vastuulla, kehitysaikainen testaaminen on ensiarvoisen tärkeätä. Tästä johtuen tuotteen tulee tukea kattavasti testaajan ja automaatiotestien musta laatikko -testausta, sekä kehittäjän tekemää kehitysaikaista testausta.

Tuotteen käyttöympäristö ja loppukäyttäjän henkilökunta määrittää, miten tuotetta käytetään ja miten tärkeään asemaan luotettavuus ja vikasietoisuus nousevat. Loppuasiakkaan oletetun toimintamallin perusteella huoltohenkilökunta ei ole välttämättä tavoitettavissa pitkiin aikoihin ohjelmiston vikaantumisen tapahtuessa. Tästä johtuen ohjelmiston tulee olla mahdollisimman vikasietoinen ja toipua itsestään suurimmasta osasta virhetilanteista. Pääohjelman tulee pystyä vastaamaan lukuisiin virhetilanteisiin, joista pääohelman aiheuttamia voivat olla mm. lukkiutuminen, tai levytilan tai muistin loppuminen. Mikäli pääohjelma kaatuu, tai jokin säie lukkiutuu, ohjelman tulee nopeasti käynnistyä uudelleen automaattisesti ilman käyttäjän toimenpiteitä. Tätä tehtävää varten tarvitaan vahtikoira ja uudelleenkäynnistys-mekanismi.

3.3 Arkkitehtuurin kuvaus

Arkkitehtuurin kuvaamiseen on olemassa lukuisia eri menetelmiä, joilla esitetään arkkitehtuurin toiminnallisuus kokonaisuudessaan. Näihin menetelmiin lukeutuu muun muassa *4 + 1 architectural view model*, malli, jossa esitetään arkkitehtuuri kokonaisuutena neljän näkymän ja skenaarioiden avulla. Tässä luvussa on tarkoitus kuvata arkkitehtuuri mukaillen näitä eri malleja, pitäen teksti kutenkin kompaktina ja johdonmukaisena. Myöhemmin aliluvun 4.4 skenaarioissa täydennetään tämän aliluvun kuvausta niissä määrin, että skenaariot ovat ymmärrettävissä.

Tämä aliluku esittelee useita eri näkymiä ja konsepteja, jotka ovat olennaisia arkkitehtuurin ymmärtämisen kannalta. Ensin aliluvuissa 3.3.1 ja 3.3.2 käydään läpi yleiskuvaus ja laitteisto, missä ohjelmistoa suoritetaan. Tämän jälkeen aliluvussa 3.3.3 esitellään moduuleihin liittyvät konseptit ja aliluvussa 3.3.4 moduulien muodostama kokonaisuus, mitkä kattavat oleellisimman osan tätä lukua. Myöhemmin aliluvuissa 3.3.5 ja 3.3.6 täydennetään esiteltyä arkkitehtuuria vahtikoiran ja toiminnallisten esimerkkien osalta.

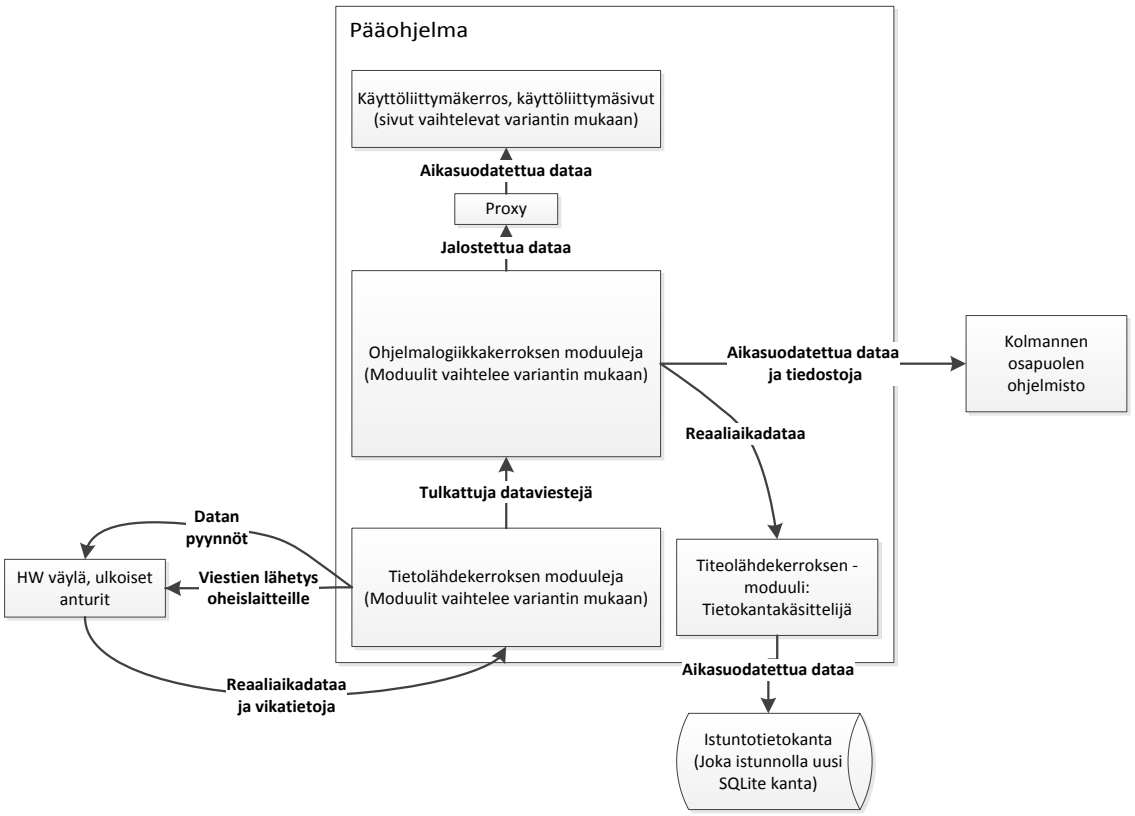
3.3.1 Yleiskuvaus

Ohjelmistolle asetetut vaatimukset, ohjelmiston tehtävät ja asiakkaan liiketoimintanäkökulmat ovat ohjanneet tarkkaan, missä rajoissa arkkitehtuuria on voitu kehittää. On käynyt ilmi, että tärkeimpiin laadullisiin ominaisuuksiin lukeutuvat mm. varioitavuus, modulaarisuus, suorituskyky ja luotettavuus. Ohjelmiston toiminta on hyvin samanlaista suurimman osan toiminnallisen määrittelyn vaatimuksien tapauksessa. Ohjelmisto koostuu pääosin toiminnoista, jotka on jaoteltavissa seuraaviin vaiheisiin:

- datan vastaanotto oheislaitteilta
- datan tulkkaaminen
- tulkatun datan jalostaminen yksinkertaisella laskennalla, jos tarve
- jalostetun datan esittäminen käyttöliittymällä
- jalostetun datan tallentaminen tietokantaan, jos vaatimus
- jalostetun datan välittäminen RT API:lle, jos vaatimus.

Datan kulku on pääosin yksisuuntaista oheislaitteilta pääohjelmalle. Oheislaitteiden lähettämät viestit ovat luonteeltaan pääosin yksittäisiä tietueita, jotka kertovat mm. anturiarvoja tai muita laitetietoja. Suurin osa näistä viesteistä ajetaan konfiguroitavan viestitulkin läpi, joka jäsentää viesteistä oleelliset tietueet. Suurin osa parsituista viesteistä on valmiissa muodossa näytöllä näytettäväksi ja tietokantaan tallennettavaksi. Pieni osa viesteistä välitetään monimutkaisemmalle logiikalle, jotka ovat tilakoneita, laskureita tai näiden yhdistelmiä. Sekä viestien yksinkertaisuus että niiden käsittelyn suoraviivaisuus on ollut helpottava tekijä arkkitehtuurin suunnittelun kannalta.

Arkkitehtuuri on jaettu kolmeen kerrokseen: tietolähdekerrokseeen (engl. data provider layer), ohjelmalogiikkakerrokseen (engl. application layer) ja käyttöliittymäkerrokseen (engl. presentation layer). Alimman kerroksen, eli tietolähdekerroksen päätehtävänä on hoitaa kaksisuuntaista kommunikointia oheislaitteiden kanssa sekä tulkata oheislaitteilta vastaanotettuja viestejä ohjelmalogiikkaa varten. Ohjelmalogiikkakerroksen moduulien tehtävänä on käyttää tietolähdekerroksesta vastaanotettuja viestejä monimutkaisemmassa laskennassa ja tilakoneiden ylläpitämisessä. Ylin kerros, käyttöliittymäkerros, pitää sisällänsä vain datan esittämiseen liittyvää koodia. Kerrosten suunnittelussa on otettu huomioon se, että esimerkiksi käyttöliittymäkerroksen variointi tai toteuttaminen uudelleen ei vaadi mitään muutoksia alempiin kerroksiin ja että koko käyttöliittymäkerroksen poistaminen ei vaikuta alempien kerrosten toimintaan. Kerrosjaottelu ja datan eteneminen on nähtävissä kontekstinäkymästä (kuva 6).



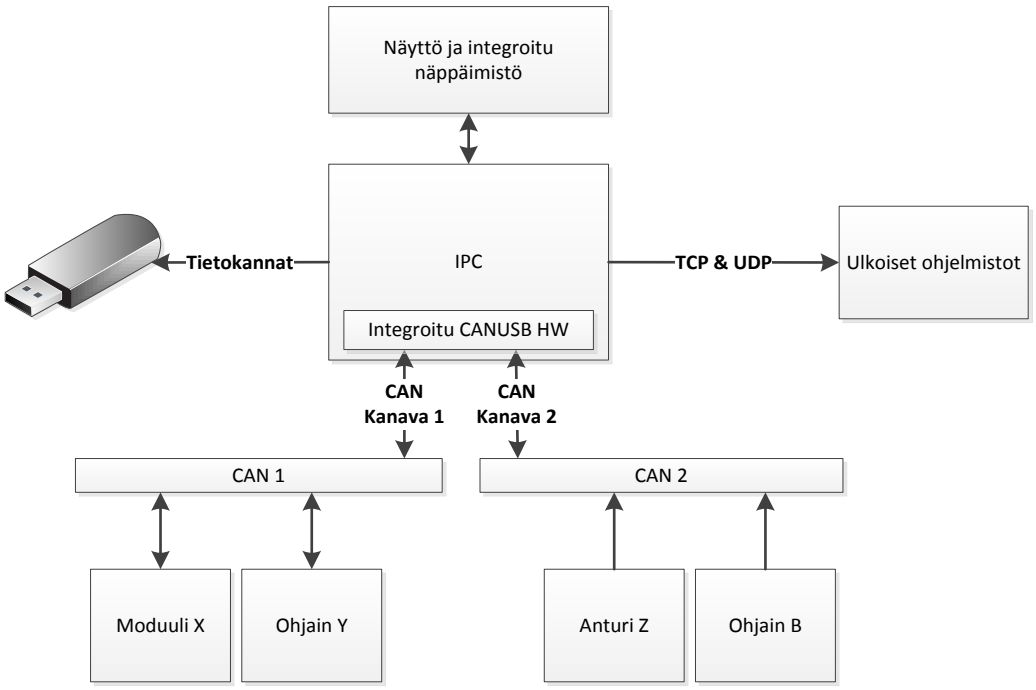
Kuva 6 Yksinkertaistettu kontekstinäkymä pääohjelman päätoiminnallisuuksista

Arkkitehtuuriin kuuluva reaaliaikarajapinta on osa ohjelmalogiikkakerrosta, mikä tarjoaa reaaliaikadataa ja pääohjelman tallentamia tiedostoja ulkoisille ohjelmistoille. Reaaliaikarajapinta tarjoaa lähiverkon yli samaa dataa, jota näytetään näytöllä ja tallennetaan tietokantaan. Tätä rajapintaa käyttämällä kolmannen osapuolen ohjelmisto voi toteuttaa erillisen käyttöliittymäkerrosta vastaavan toteutuksen. RT API:n yhteydessä on myös huoltorajapinta, jonka avulla pääohjelmalta voi ladata SQLite-tietokantoja tai muita pääohjelman tiedostoja. Huoltorajapinta on kaksisuuntainen ja sen avulla voi päivittää ja muokata tiettyjä pääohjelman osia. Huoltorajapinnan vaatima toiminnallisuus on osaltaan monimutkaistanut joidenkin moduulien toteutusta, mutta arkkitehtuuriin ytimeen sillä ei ole ollut kovin mainittavaa vaikutusta.

3.3.2 Laitetason näkymä

Analysoitava tuote on kokonaisuus, johon kuuluu sekä tietynlainen pääohjelmalle räätälöity laitteisto että laitteistolla suoritettava ohjelmakoodi. Laitteisto vastaa teknisiltä tiedoiltaan suurilta osin tavallista pöytätietokonetta, mutta suurimmat eroavaisuudet ovat fyysisessä koteloinnissa ja laitteistorajapinnoissa. Tietokoneessa on suora laitteistotuki mm. CAN- ja sarjaporttiväylille, joista tulee kaikki pääohjelman vastaanottama ja käsittelemä reaaliaikainen informaatio. Eri tuotekonfiguraatoissa on toisistaan poikkeavat ohjelmistot, joita varten on räätälöity erilainen laitteisto ja siten väyläkonfiguraatio voi olla eri tuotteissa erilainen. Yhteistä kaikilla tuotteilla on kuitenkin se, että suurin osa tiedoista vastaanotetaan CAN-väylistä ja muut väylät voivat osasta tuotteita puuttua.

Kuva 7 esittää esimerkkituotetta, johon on kytketty kahden CAN-väylään liitetyt oheislaitteet. Molemmissa CAN-väylissä on useita oheislaitteita, joista kukin tuottaa useita eri viestikehyksiä. Tietokone vastaanottaa kymmeniä erilaisia viestikehyksiä, joista vain osa on pääohjelman kannalta kiinnostavia viestejä, ja loput ovat ylimääräistä informaatiota, jota ei käsitellä. Kommunikointi CAN-väylään on pääosin yksisuuntaista: pääohjelma seuraa CAN-liikenteestä niitä viestejä, joita oheislaitteet lähettävät säännöllisillä intervalleilla. Osa vastaanotetuista viesteistä vaatii erillisen pyyntöviestin lähettämistä väylälle ja monesta CAN-kehyksestä koostuvat viestit (esim. KWP2000) vaativat monimutkaisempaa kaksisuuntaista kommunikointia.

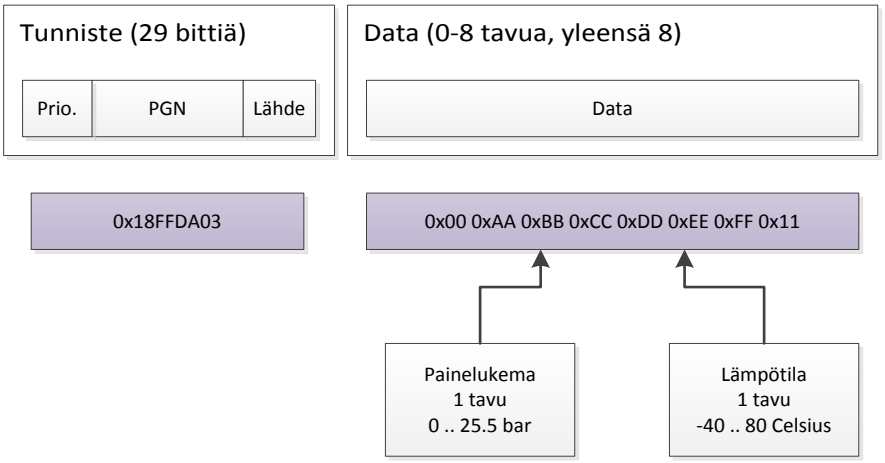


Kuva 7 Laitetason näkymä esimerkkituotteesta

Arkkitehtuurin kannalta oleellista tietoa CAN-viestin rakenteesta on vain ne osuudet, jotka välittyvät kirjastolta sovellukselle. SAE J1939-standardin mukainen CAN-toteutus rakentuu hieman alle nelitavuisesta, 29-bittisestä tunnisteesta ja maksimissaan kahdeksantavuisesta datasisällöstä. Väylällä kulkee enemmänkin datakenttiä, jotka ovat tarpeellisia OSI-mallin (engl. Open System Interconnection Reference Model) alemmilla kerroksilla ja jotka abstrahoituvat pääohjelmalta piiloon. Sovellukselle menevistä tiedoista tunniste (engl. frame identifier) yksilöi viestit ja sen avulla sovellus voi tietää datasisällön tarkan rakenteen. Datasisältö vuorostaan noudattaa aina samaa, tunnisteella yksilöitävää rakennetta.

Kuva 8 pitää sisällänsä esimerkkiviestin tunnisteella 0x18FFDA03. Viestin lähettäjän rajapintaspesifikaatioista käy ilmi, että PGN-numero 0xFFDA tulevissa viesteissä on anturin painelukema kolmannessa tavussa ja lämpötilatieto kuudennessa tavussa. Tunnisteen viimeinen tavu on lähettäjän ID, jolloin useampi eri moduuli voi lähettää samaa kehystä samalla PGN-numerolla ja vastaanottajan on edelleen mahdollista

yksilöidä viestin lähde. Esimerkkikuvassa lähde 0x03 lähettää viestin 0xFFDA ja prioriteettitavu 0x18 tarkoittaa suurinta prioriteettia 6.



Kuva 8 Sovelluskehittäjälle näkyvä osa SAE J1939 protokollanmukaisen pidennetyn CAN viestin rakenteesta

Laitteeseen on mahdollista kytkeä myös ulkoisia ohjelmistoja ethernet-liitännällä sekä USB-muistitikku huoltotoimia varten. Ulkoiset ohjelmistot ovat tuotteen käyttökohteesta riippuen joko jatkuvasti tai harvoin yhteydessä. Lähiverkkoon kytketyillä huolto- ja monitorointiohjelmistoilla on mahdollista vastaanottaa reaaliaikaisesti sitä dataa, jota pääohjelma vastaanottaa oheislaitteilta ja pääohjelman jalostamaa dataa. Reaaliaikadatan vastaanoton lisäksi huolto-ohjelmistoilla on mahdollista päivittää huoltotietoja, tuotetietoja, yhteensopivuustietoja ja vastaanottaa historiadataa XML-tiedoston ja SQLite-tietokantojen muodossa. Konfiguraatio-tiedoston ja tietokantojen siirrot voidaan tehdä myös pääohjelmasta käsin käyttäen USB-muistitikkua siirtomediana.

3.3.3 Moduulit

Analysoitavan ohjelmiston arkkitehtuuri on hyvin modulaarinen ja kaikki varioitavat toiminnallisuudet on jaettu yksittäisiin moduuleihin, jotka on helppo lisätä tuotteen käännökseen mukaan. Pääohjelma koostuu useista kymmenistä moduuleista ja suurimmassa tuotevariantissa niitä on yhteensä 47 kappaletta. Arkkitehtuurissa on kaikille tuotteille yhteinen ydinkoodi, joka pitää sisällänsä kaikkien varianttien tarvitsemia perustoiminnallisuuksia ja koodia, jota kaikki moduulit tarvitsevat, kuten viestinvälitys ja lokalisointi. Variant-kohtaiset moduulit ovat pienempiä kokonaisuuksia, jotka toteuttavat yhden loogisen toiminnallisuuskokonaisuuden ja jotka on mahdollista lisätä tuotteeseen hyvin marginaalisilla koodimuutoksilla. Tässä aliluvussa käsitellään variant-kohtaisten moduulien toimintaan ja kommunikointiin liittyvät arkkitehtuuriratkaisut, suunnittelusäännöt ja yleiskuvaus.

Moduulien suunnittelusäännöt

Arkkitehtuurin oleellisesti tärkeimmistä vaatimuksista on hyvä varioitavuus, mikä on vaikuttanut moduulien suunnittelusääntöihin. Varioitavuutta on pyritty parantamaan sillä, että moduulit olisivat mahdollisimman riippumattomia siitä, mitä muita moduuleja

on ladattu ajoon, tai siitä, mistä ja missä muodossa data alkujaan tulee. Moduulien kokoon ja niissä olevaan koodiin vaikuttaa myös asiakkaan vaatimus, että missään variantissa ei saisi olla sinne kuulumatonta koodia eikä koodia, jota ei ajeta. Tämä johtaa siihen, että suurin osa moduuleista on pieniä ja toiminnaltaan yksinkertaisia. Moni moduuli vastaa vain yhdestä tai kahdesta toiminnosta, joita ovat mm. tilakone, tilan määritys tai kommunikointi yhden oheislaitteen kanssa. Lisäksi suurin osa moduuleista on konfiguroitavissa erillisillä XML-tiedostoilla, jotta moduulien toiminnallisuutta on mahdollista muuttaa tuotekohtaisesti.

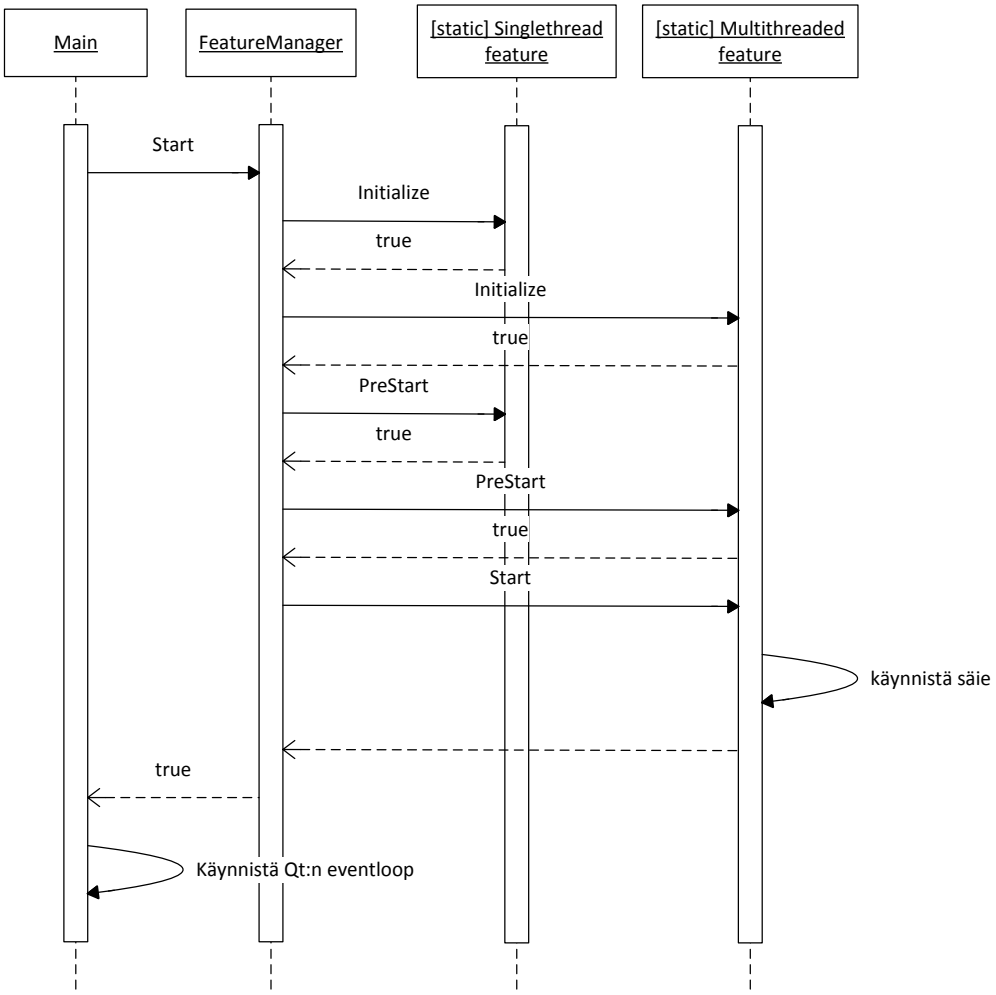
Moduulien joukossa on myös muutama erityisen tärkeä, konfiguroitavissa oleva moduuli, jotka ovat mukana jokaisessa tuotteessa. Näistä tärkeimpiä ovat mm. CAN-viestien tulkkaaja (DataMessageParser-moduuli), kerrosten välisten viestien välittäjä (MessageForwarder-moduuli) ja tietokantamoduuli. Nämä ovat esimerkkejä moduuleista, jossa suurin osa toiminnallisuudesta on määritelty konfiguraatio-tiedostoilla. Näissä tapauksissa lähdekoodi on sama eri tuotevarianteilla ja toiminnalliset erot saavutetaan konfiguroimalla.

Moduulien lataus

Kaikki ladattavat moduulit käyvät käynnistyksessä samat ennalta määritetyt vaiheet läpi, mikä helpottaa moduulisuunnittelua ja moduulien keskinäistä synkronisoimista. Jokaisesta moduulista luodaan staattinen instanssi heti pääohjelman käynnistyessä, mutta moduulit ajavat vain lähes tyhjät rakentajansa. Moduulien käynnistysrutiinit tehdään moduulien kantaluokan rakentajassa, joka rekisteröi moduulit automaattisesti *FeatureManager*-instanssille, joka myöhemmin ohjaa moduulit käyntiin.

Moduulit käyvät käynnistyksen yhteydessä kaksi eri esikäynnistysvaihetta: alustus (*Initialize-funktio)* ja esikäynnistys (*PreStart-funktio)*. Alustus on ensimmäinen vaihe, jossa moduulit tekevät kaikki tarvittavat toimenpiteet, jota ne käynnistyksessä tarvitsevat, kuten konfiguraatiotiedostojen lukeminen tai valmistautuminen tulevia viestejä varten. Moduulit eivät tiedä muiden moduulien tiloja, joten mitään kommunikaatiota ei vielä alustusvaiheessa tehdä. Alustusvaiheen jälkeen suurin osa moduuleista on toimintakunnossa ja valmiina käsittelemään viestejä.

Kun kaikki moduulit on alustettu, aloitetaan esikäynnistysvaihe. Esikäynnistysvaihe on toinen alustusvaihe, jossa moduulit voivat asettaa alkuarvonsa sen mukaan, mitä muut moduulit viestivät. Tässä vaiheessa muun muassa historiatiedoista vastaava moduuli lähettää edellisen suorituskerran viimeiset lukuarvot, jotta laskurimoduulit voivat alustaa laskurinsa oikeisiin lukemiin. Esikäynnistyksen jälkeen voidaan olettaa, että kaikki moduulit ovat täydessä suoritusvalmiudessa, mutta oheislaitedataa tarjoavat säikeistetytmoduulit eivät vielä ole ajossa. Esikäynnistyksen jälkeen säikeistetyt moduulit ajetaan käyntiin ja pääohjelma aloittaa normaalin suoritusrutiininsa. Alla olevassa tapahtumasekvenssikaaviossa (kuva 9) on esimerkki ohjelman käynnistyksen vaiheista.



Kuva 9 Moduulien latausvaiheet

Tapahtumasekvenssikaavio esittää ohjelman käynnistysvaiheet kahden moduulin osalta, kun pääohjelma ajetaan eri käynnistysvaiheiden läpi. Moduulit on esitetetty kaaviossa nimekkeillä *Singlethread* ja *Multithreaded*, millä viitataan siihen suoritetaanko viestinkäsittely synkronisesti vai asynkronisesti. *Singlethread*-moduulin tapauksessa moduuli on suorituksessa ainoastaan silloin, kun se vastaanottaa viestin, tai kun jokin moduulin sisäinen ajastin tuottaa Qt-signaalin. *Multithreaded*-moduuli eroaa siten, että se suorittaa omaa Start-funktiotaan omassa säikeessään. Säikeistetty moduuli voisi olla tässä tapauksessa CAN-datan vastaanottomoduuli ja säikeistämätön voisi olla CAN-datan tulkkausmoduuli.

Moduulien kommunikointi

Moduulit kommunikoivat viestinvälitysarkkitehtuurin mukaisesti, jolloin yksikään moduuli ei suoraan kutsu toisen moduulin funktioita, vaan kommunikointi tapahtuu viestien välityksellä. Viestien välityksestä vastaa *MessageHandler*-viestinvälittäjä-instanssi, joka vastaanottaa viestejä miltä tahansa lähteeltä ja välittää ne kyseisistä viesteistä kiinnostuneille vastaanottajille. Viestit yksilöidään niille annetun

hierarkiatunnisteen (Hierarchy-luokka) perusteella, jonka sekä lähettäjä että vastaanottaja tietää. Jokaisen lähetettävän viestin yhteydessä annetaan hierarkiatunniste, minkä MessageHandler välittää niille moduuleille, jotka ovat ilmoittaneet haluavansa vastaanottaa kyseisiä viestejä. Viestinvälittäjä voi välittää useaa eri viestityyppiä, jotka poikkeavat datasisältöjensä rakenteelta. Välitetyt viestit ovat pääasiallisesti hyvin pienikokoisia, sillä ne pitävät sisällänsä pääosin yksittäisiä lukuarvoja ja siihen liittyvää metadataa.

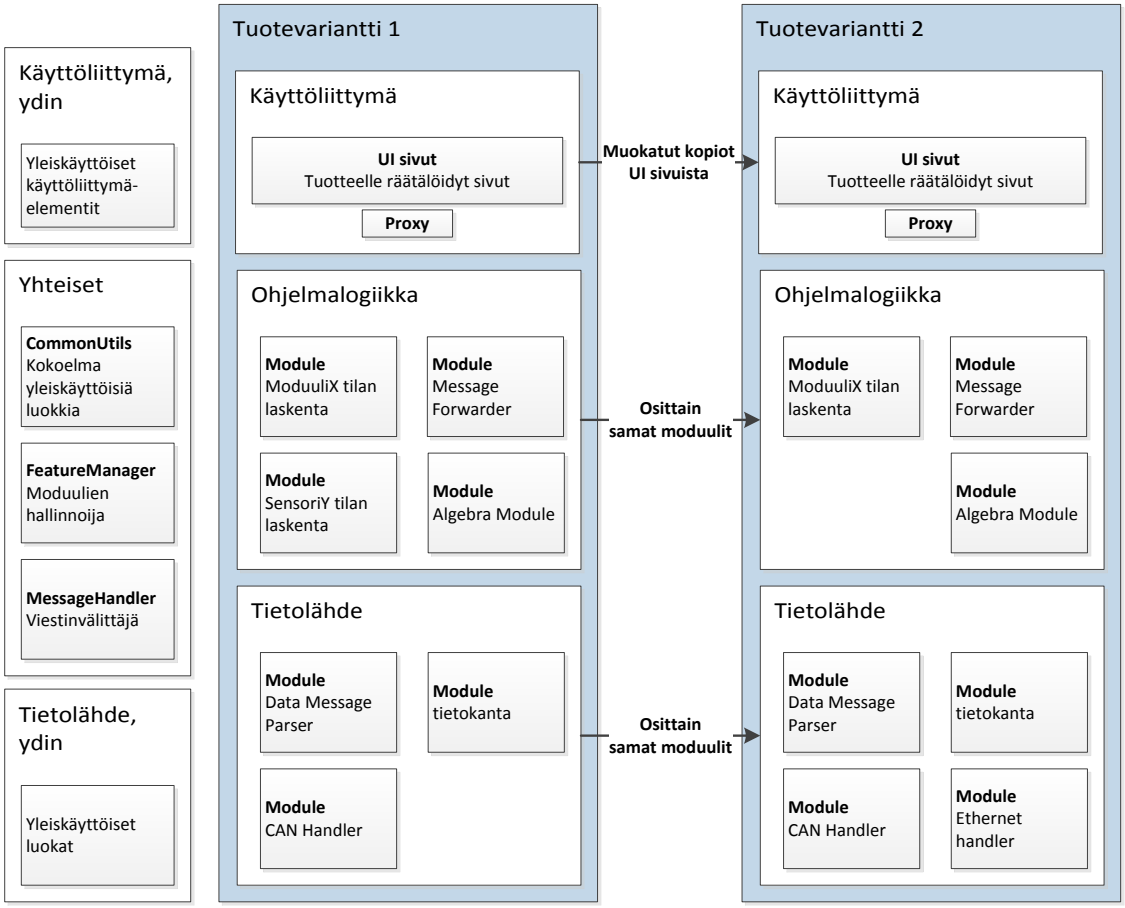
Hierarkiatunniste on merkkijono, joka koostuu pisteillä erotetuista hierarkiatasoista. Esimerkiksi tietolähdekerrokselta lähetetty *AnturiX*:n painetieto voisi saada hierarkian: ”Data.AnturiX.Paine”. Mikäli moduuli on rekisteröinyt vastaanottamaan hierarkiaa ”Data.AnturiX”, se saa kaikki viestit, joiden hierarkiatunnisteet alkavat tai vastaavat täysin kyseistä hierarkiaa. Tällä periaatteella mikä tahansa moduuli voisi vastaanottaa kaikki tietolähdekerroksen viestit rekisteröimällä hierarkian ”Data”, tai vastaavasti logiikkakerroksen viestit hierarkialla ”Application”.

Jotta viestit voitaisiin välittää oikeille moduuleille, moduulien tulee etukäteen rekisteröidä viestinvälttäjälle ne hierarkiat, joihin kuuluvat viestit ne haluavat vastaanottaa. Kukin moduuli voi vastaanottaa rajattoman määrän eri hierarkioita ja jokaisen välitetyn viestin yhteydessä vastaanottaja saa tiedon, mitä hierarkiaa kyseinen viesti vastaa. Jokainen lähetetty viesti voidaan myös välittää rajattomalle määrälle eri vastaanottajia. Jokainen moduuli myös voi myös käyttää rajattoman määrän erilaisia hierarkioita lähettäessään viestejä. Tästä johtuen moduulien välisten viestiyhteyksien luominen on hyvin joustavaa.

Koska moduulit eivät voi vastaanottaa viestejä ennen rekisteröintiä, moduulien käynnistysvaiheissa on huomioitu, että viestejä ei lähetetä ennen kuin kaikki moduulit ovat tehneet rekisteröitymisensä. Edellä kohdassa ”Moduulien lataus” esitetyssä alustusvaiheessa moduulit rekisteröivät kaikki hierarkiat, jota ne haluavat vastaanottaa. Esikäynnistysvaihe alkaa heti kun kaikki moduulit on alustettu, jolloin yksikään lähetetty viesti ei jää välittämättä sitä tarvitseville vastaanottajille. Kun ohjelma on lähtenyt käyntiin ja oheislaitteilta on alettu vastaanottamaan dataa, viestejä kulkee jatkuvasti hyvin vilkasta tahtia moduulien välillä. Kommunikoinnin esimerkkitapauksia on esitetty lisää aliluvussa 3.3.6.

3.3.4 Arkkitehtuuri kokonaisuutena

Pääohjelman arkkitehtuuri on jaettu kolmeen kerrokseen, joista alemmat kaksi kerrosta muodostuvat pelkästään varioitavista moduuleista. Ohjelman kaikki toiminnallisuus tapahtuu vain moduuleissa, joten kaikki käyttöliittymän ulkopuoliset ominaisuudet voidaan määrittää sillä, mitä moduuleja ohjelmaan tulee mukaan ja miten ne on konfigurioitu. Tuotteesta on mahdollista ottaa kaikki moduulit pois, jolloin ohjelmaan jää kaikille tuotteille yhteinen arkkitehtuurin ydin ja tyhjää dataa näyttävä käyttöliittymä. Kuva 10 esittää esimerkkitapauksen kahdesta eri tuotevariaatiosta, missä on hieman eri toiminnallisuudet.



Kuva 10 Järjestelmän rakentuminen eri tuotteiden osalta

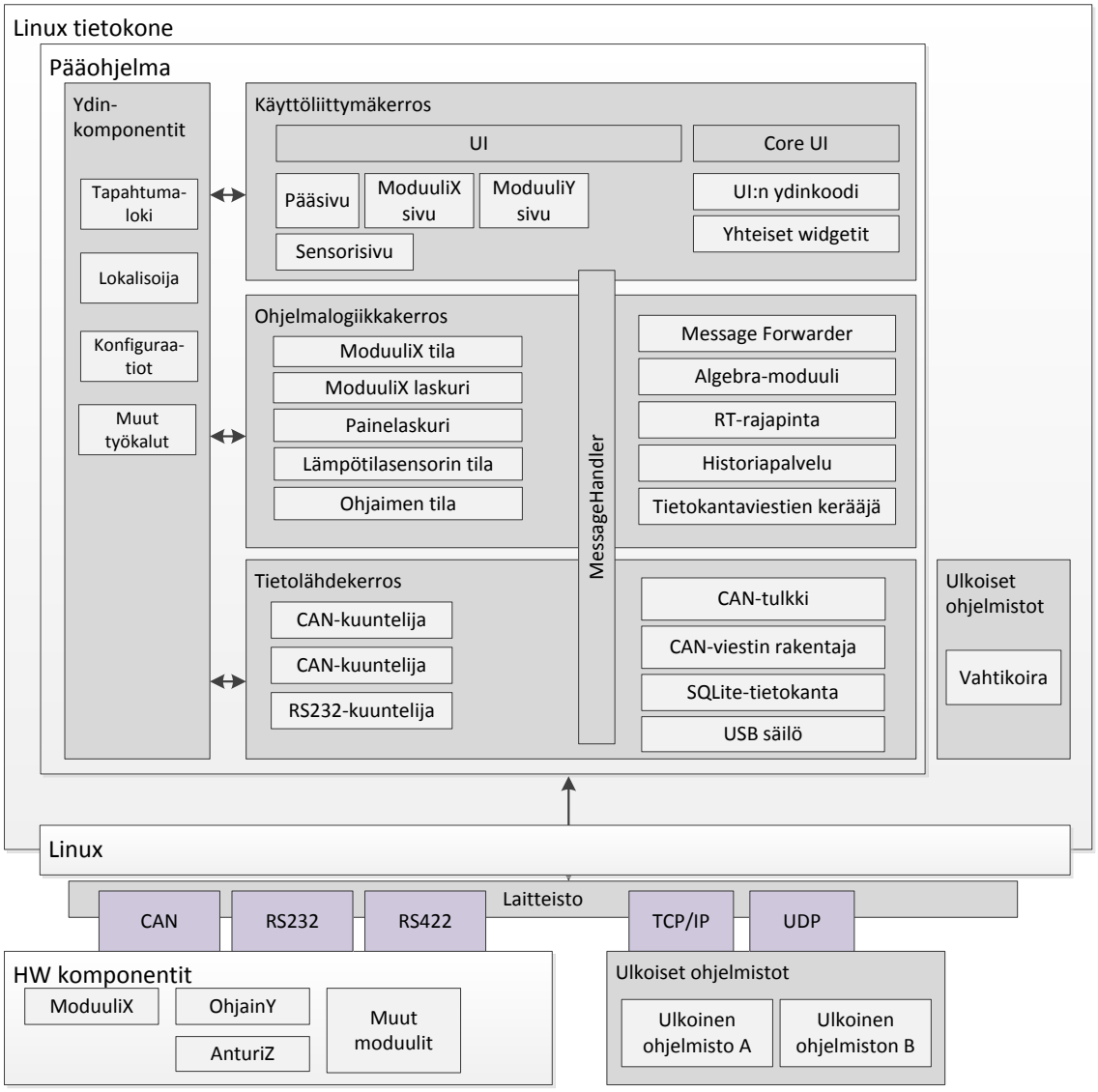
Käyttöliittymäkerroksen osalta varioitavuus ei ole samalla tasolla kuin ohjelman muissa osissa. Uutta tuotetta tehtäessä käyttöliittymän koodit kloonataan aikaisemmasta tuotteesta ja sitä modifioidaan niiltä osin, mitä on tarve. Käyttöliittymäkerroksessa on kuitenkin suuri määrä kaikille tuotteille yhteistä koodia, mikä pienentää huomattavasti kloonattava koodin määrää. Tuotekohtaisissa käyttöliittymäkoodeissa on pääosin hyvin minimaalinen määrä koodia, kuten sensorisivuilla on vain muutaman rivin listaus näytettävistä sensoridatoista. Tällä lähestymistavalla on mahdollistettu se, että käyttöliittymään on mahdollista tehdä varianttikohtaisesti huomattavia kosmeettisia muutoksia.

Yksikään tuotteeseen tuleva moduuli ei voi tietää tarkalleen, mistä sen vastaanottama data tulee, tai mihin sen lähettämä data menee. Jotkin yleiskäyttöiset moduulit voivat generoida pienen määrän ylimääräistä dataa, jota mikään moduuli ei käytä. Samaten joillain moduuleilla voi olla pieni määrä ylimääräisiä laskureita, jotka eivät vastaanota mitään dataa, koska kyseisessä varintissa juuri niitä viestejä ei tuoteta. Tämä riippumattomuus moduulien välillä mahdollistaa sen, että datalähde voidaan joustavasti vaihtaa toiseen. Tästä johtuen on ollut mahdollista tehdä kehitystyökalu, joka simuloi samanaikaisesti kaikkia ulkoisia rajapintoja tuottamalla niiltä odotettuja viestejä.

Se, että moduulit ovat toisistaan riippumattomia ja viestit voivat kulkea vapaasti kerrosten välillä tarkoittaa sitä, että koodin näkökulmasta ei oikeasti ole rajavetoa tietolähde- ja ohjelmalogiikkakerrosten välillä. Todellisuudessa käyttöliittymäkerrosta

alemmat kerrokset koostuvat samanlaisista moduuleista, joiden ainoat erot ovat kansiorakenteessa ja siinä mihin kohtaan arkkitehtuurikuvaa ne on piirretty. Myös tietolähdekerros on edelleen jaettu moduulien toiminnan perusteella kolmeen moduuliryhmään: *laitteisto-*, *logiikka-* ja *systeemimoduulit*. Arkkitehtuurillisesti oleellista moduulijaossa on se, että ohjelmalogiikkakerroksen moduulit eivät tiedä mitään datan alkuperäisestä lähteestä tai viestinnässä käytetyistä protokollista.

Kuva 11 pitää sisällään yleiskuvan koko arkkitehtuurista moduulien esimerkkikokoonpanolla. Tietolähdekerroksen väyläkuuntelijamoduulit kuuluvat *laitteistomoduuleihin*, sillä ne ovat suorassa yhteydessä laitteiston kanssa. CAN-tulkki ja viestin rakentaja kuuluvat *logiikkamoduuleihin*, sillä ne ovat tekemisissä laitteistoprotokollien kanssa ja sisältävät enemmän logiikkaa. Tietokanta ja USB-käsittelijä kuuluvat *systeemimoduuleihin*, sillä ne ovat tekemisissä jossain määrin alla olevan käyttöjärjestelmän kanssa.



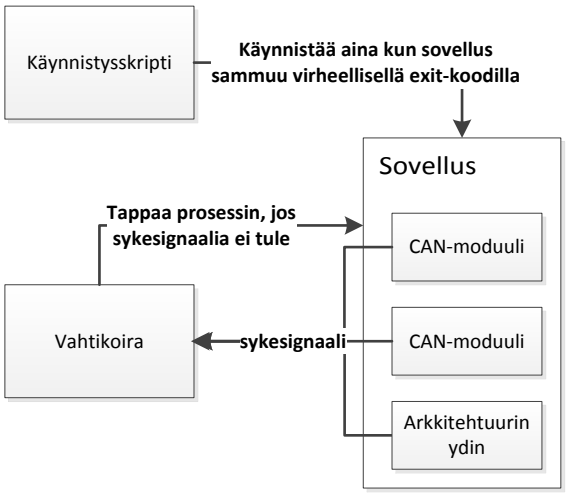
Kuva 11 Koko järjestelmän korkean tason moduulinäkymä esimerkkimoduuleilla

Keskimmäisen kerroksen moduulit eivät ole enää tekemisissä oheislaitteiden kanssa, vaan ne olettavat alemman kerroksen hoitavan laitteistoon liittyvä toimenpiteet. Ohjelmalogiikkakerros toteuttaa suurimman osan kaikesta tuotteen toiminnallisuudesta. Moni tämän kerroksen moduuleista on kaikissa tuotteissa, mutta niiden toiminnallisuus eroaa konfiguraatiotiedostojen määrittelyiden mukaan. Kaikki vastaanotettujen lukujen perusteella tehtävä tilakone- ja laskurilogiikka sekä muu monimutkainen toiminnallisuus tehdään tässä kerroksessa. Tämän kerroksen tuottama data on jalostettu niin pitkälle, että sitä voi näyttää sellaisenaan käyttöliittymällä tai tallentaa tietokantaan.

Käyttöliittymäkerros jakautuu ydinkoodiin, joka tarjoaa kehyksen käyttöliittymän toteuttamiseen, sekä tuotekohtaiseen koodiin, joa pitää sisällänsä tuotekohtaisesti räätälöidyt UI-sivut. Käyttöliittymä koostuu pääosin sivuista, jotka esittävät vastaanotettuja anturitietoja. Vain pieni osa käyttöliittymästä johtaa kaksisuuntaiseen kommunikointiin alempien kerrosten kanssa, mistä johtuen suurin osa sivujen sisällöstä on yksinkertaisia listauksia näytettävästä datasta. Suurin osa käyttöliittymäkoodista sijaitsee kaikkien tuotteiden jakamassa ydinkoodissa, joten tuotekohtaista käyttöliittymätyötä on suhteellisen vähän.

3.3.5 Vahtikoira

Ohjelman luotettavuutta on pyritty parantamaan vahtikoiramekanismilla (kuva 12), jolla varmistetaan ohjelman jatkuva saatavuus ja virheistä toipuminen. Vahtikoira on erillinen prosessi, joka seuraa jatkuvasti sovelluksen lähettämiä sykesignaaleja ja muistinkäyttöä. Mikäli jokin moduuli jättää lähettämättä useamman peräkkäisen sykesignaalin tai vapaan muistin määrä laskee kriittiselle tasolle, vahtikoira pakotetusti lopettaa pääohjelman prosessin.



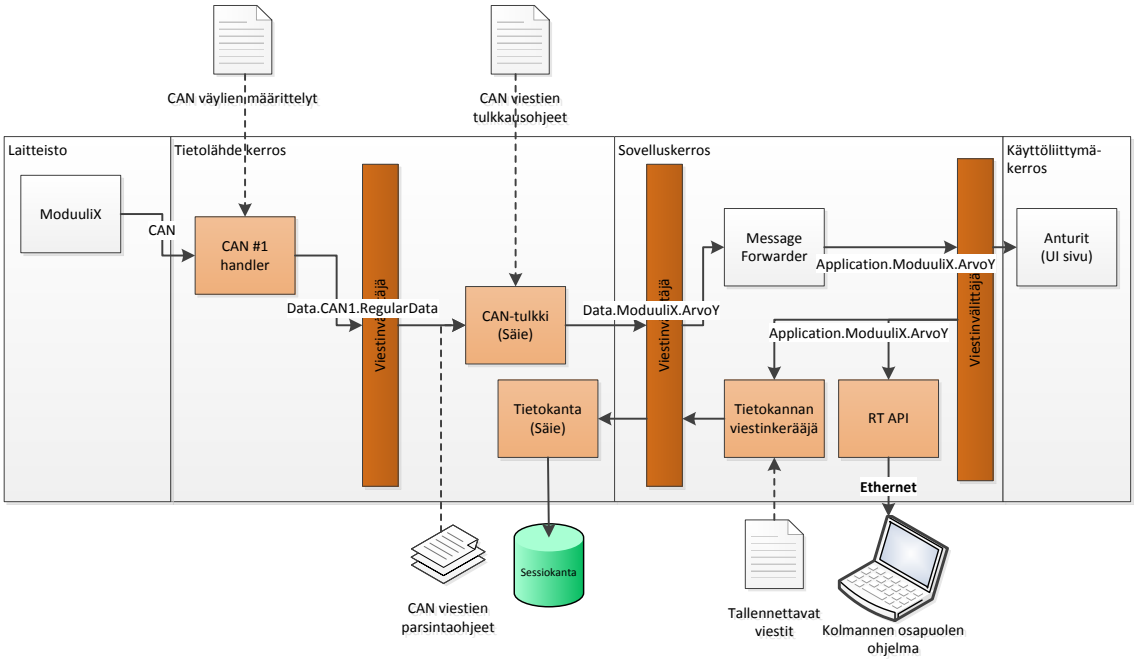
Kuva 12 Vahtikoiramekanismin toiminta normaalitilanteessa

Käyttöjärjestelmän käynnistymisen yhteydessä ajetaan skripti, jonka tehtävänä on käynnistää pääohjelma aina, kun se ei ole ajossa. Mikäli pääohjelman suoritus keskeytyy siten, että ohjelman paluuarvo indikoi virhetilannetta, skripti käynnistää pääohjelman uudelleen. Tällä varmistetaan se, että selittämättömän kaatumisen tai

vahtikoiran aiheuttaman prosessin tappamisen jälkeen pääohjelma käynnistyy välittömästi. Mikäli ohjelma sammuu palauttaen EXIT_SUCCESS-paluuarvon, käynnistysskripti ei tee enää uudelleenkäynnistystä. Tämä tapahtuu vain silloin, kun tietokone sammutetaan.

3.3.6 Datan virtaus arkkitehtuurissa

Järjestelmän sisäiselle tietovirralle on ominaista se, että tieto kulkee pääosin yhdensuuntaisesti oheislaitteilta ohjelmalogiikan läpi näytölle. Järjestelmässä etenevät viestit kulkevat samoja hierarkiatunnistekohtaisia reittejä koko ohjelman suorituksen ajan. Kuva 13 esittää esimerkin tyypillisestä skenaariosta: Viesti vastaanotetaan CAN-väylältä, jäsennetään ja välitetään käyttöliittymälle. Osa ohjelmalogiikassa käsitellyistä viesteistä annetaan käyttöliittymän lisäksi reaaliaikarajapinnalle ja tietokannalle.

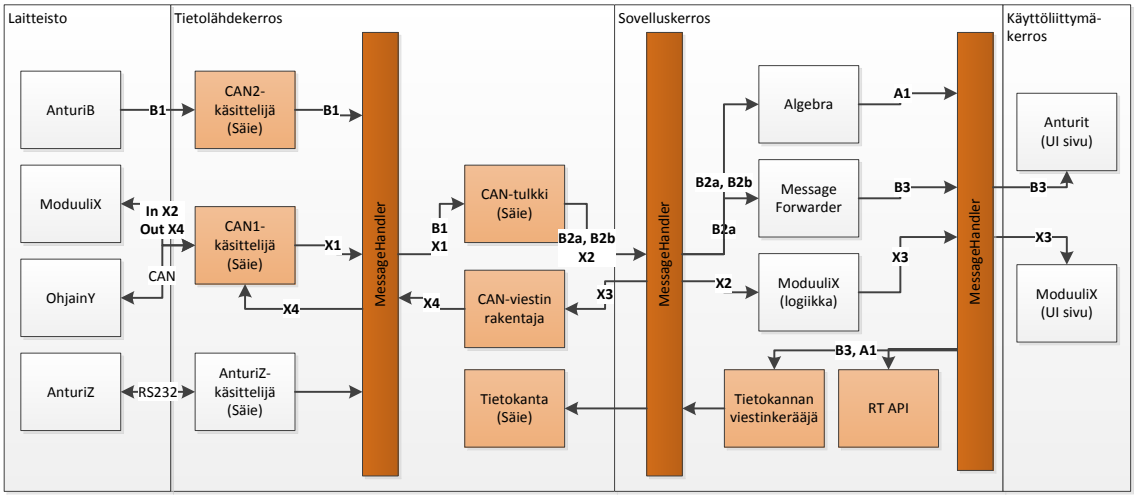


Kuva 13 Moduulin ”ModuuliX” lähettämän viestin eteneminen järjestelmässä. Viestinvälittäjiä on vain yksi instanssi.

Suurin osa järjestelmän vastaanottamista viesteistä on lähtöisin joltain järjestelmään liitetyltä J1939-protokollan CAN-väylältä. Viestit, jotka ovat sellaisenaan tulkattavissa, välitetään CAN-viestien tulkille. Sillä tulkitaan CAN viestin raakadatasta pääohjelman kannalta oleelliset viestit, muuntaa ne moduulien kannalta helpommin käytettävään muotoon ja lähettää konfiguraatioissa määritellyillä viestihierarkioilla viestinvälittäjälle. Esimerkin (kuva 13) tapauksessa CAN-väylässä olevan ModuuliX:n lähettämä viesti voisi sisältää kaksitavuisen tietueen, joka tulkitaan liukulukuna ja jonka tulkki lähettää eteenpäin hierarkialla ”Data.ModuuliX.ArvoY”. Tulkin lähettämä viesti sisältää aina seuraavat tiedot: Viestin hierarkia, datatyyppi, lukuarvo, arvon tila ja viestin saapumisaika.

Tulkin lähettämä viesti on ApplicationMessage-olio, joka on kaikkien sovelluskerroksen moduulien tunnistama muoto. Esimerkkiviesti ”ArvoY” kulkee

MessageForwarder-moduulin läpi, jonka tärkein tehtävä tämän viestin tapauksessa on vaihtaa viestihierarkiaa Data-alkuisesta Application-alkuiseksi, jolloin viesti siirtyy teoriassa tietolähdekerrokselta ohjelmalogiikkakerrokselle. Tämän hierarkiamuutoksen jälkeen viesti jatkaa kulkuaan datasisällöltään muuttumattomana käyttöliittymälle, reaaliaikarajapinnalle ja tietokannalle. Viestin datasisältö pysyy tässä tapauksessa samana tulkilta käyttöliittymään asti. Viestin datasisällössä oleva arvon tila voi olla validi, N/A (engl. Not Applicable, ei käytettävissä oleva), varoitus tai virhe riippuen siitä, onko arvolle asetettu oikeellisuus- tai virherajoja. Toisaalta mikäli CAN-viestiä ei ole vastaanotettu konfigurioituun aikaikkunaan, jonka perusarvo on 750 ms, CAN-tulkki lähettää viestin N/A-tilalla, jotta bisneslogiikan moduulit, käyttöliittymä ja reaaliaikarajapinnassa olevat oheislaitteet tietävät viestin vanhentuneen.



Kuva 14 Muutaman esimerkkiviestin eteneminen järjestelmässä.

Kuva 14 näyttää esimerkkitilanteen, jossa on hieman enemmän moduuleja ja viestejä. Kuvassa viestit ovat yksinkertaistettu yksittäisiksi aakkosiksi, joiden perässä on numero, joka viittaa muutoksiin joko viestin hierarkiassa tai datasisällössä. Lyhyt kuvaus AnturiB:n lähettämästä viestistä ’B’ ja siihen liittyvistä muista viesteistä:

1. AnturiB lähettää viestin ’B1’, jonka CAN2-käsittelijä lukee
2. CAN2-käsittelijä lähettää viestin ’B1’ sisäiseen viestinvälitykseen
3. ’B1’ CAN-viesti pitää sisällänsä kaksi tulkattavaa tietuetta, jotka tulkataan viesteiksi ’B2a’ ja ’B2b’.
4. Bisneslogiikassa MessageForwarder on kiinnostunut vain ’B2a’-viestistä ja välittää sen uudella hierarkialla ’B3’.
    I. Käyttäliittymä vastaanottaa tämän viestin ja näyttää käyttäjälle.
    II. Tietokantakäsittelijä vastaanottaa tämän viestin ja tallentaa tietokantaan.
    III. Reaaliaikarajapinta vastaanottaa tämän viestin ja välittää oheislaitteille.
5. Algebramoduuli on kiinnostunut molemmista, ’B2a’ ja ’B2b’ viesteistä, ja käyttää niitä laskuissaan muodostaen uuden viestin ’A1’.
    I. Tietokantakäsittelijä vastaanottaa tämän viestin ja tallentaa tietokantaan.
    II. Reaaliaikarajapinta vastaanottaa tämän viestin ja välittää oheislaitteille.

Viesti ’B’ on samalla sekä yksinkertainen että yleisin esimerkki siitä, miten viestit etenevät ja jalostuvat arkkitehtuurin sisällä. Kuvassa on esitetty myös harvinaisempi tapaus viestillä ’X’, missä vastaanotetun viestin jälkeen uusi viesti lähetetään takaisin CAN-väylälle. Esimerkit ’B’ ja ’X’ esittävät, kuinka suoraviivaista arkkitehtuurin sisäinen viestien käsittely- ja välitysmekanismi on. Toteutetussa järjestelmässä on kymmeniä moduuleja ja osa moduuleista voi muodostaa useammasta viestistä riippuvia tilakoneita, mutta viestinvälityksen kannalta moduulit käyttäytyvät hyvin samalla tavalla.

3.4 Varioinnin vaikutus arkkitehtuuriin

Arkkitehtuurin tärkein ominaisuus on hyvä varioitavuus, mikä osaltaan korostaa modulaarisuuden tärkeyttä. Jotta tuotteesta olisi mahdollista tehdä uusia variantteja pienellä työmäärällä, moduulien väliset suorat riippuvuudet eivät ole monessa tapauksessa sallittua. Siitä johtuen moduuleilla on tarve pystyä viestimään konfiguroidusti ja epäsuorasti muiden moduulien kanssa. Tämä on johtanut viestinvälitysarkkitehtuurin ja hierarkiapohjaisen viestiliikenteen kehittämiseen.

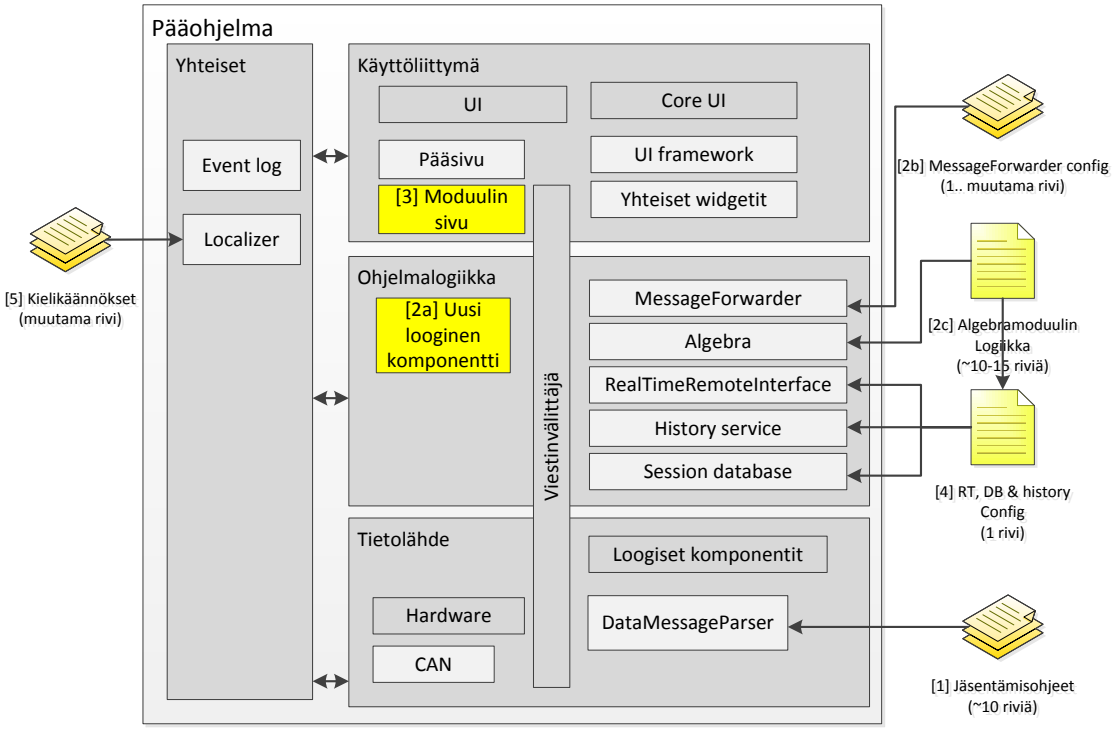
Jouhevan tuotekehityksen takia arkkitehtuurin tulee tukea automaattista kääntö- ja testausjärjestelmää jokaisen tuotevariaation kehityksessä. Tämän saavuttamiseksi ohjelman koodit on jaettu loogiseen kansiorakenteeseen, jossa kukin moduuli tulee omaan alikansioonsa. Tällöin ennen kääntämistä ajettavien skriptien on mahdollista ottaa vain tuotteeseen tulevat kooditiedostot ja generoida kääntöjärjestelmäspesifit projektitiedostot siten, että ohjelmistosta saadaan käännettyä tuotantoversiot. Jotta moduulit olisivat kansiokohtaisia kokonaisuuksia, moduuleilla ei voi olla keskenään suoria riippuvuuksia. Ainoat suora riippuvuudet voivat olla vain arkkitehtuurin ydinkoodiin, mikä tulee kaikkiin tuotevariantteihin.

Tuotekohtaisten varianttimuutosten tekeminen käy parhaiten ilmi esimerkkiskenaarion avulla. Yleisin skenaario on datan vastaanotto CAN-väylältä, sen käsittely ja välittäminen käyttöliittymälle, johon liittyvät muutokset on helppo toteuttaa nykyisellä arkkitehtuurilla ja moduulikokoonpanolla. Kuva 15 esittää skenaarion, jossa lisätään uuden vastaanotetun CAN-viestin jäsentäminen. Oleelliset muutospaikat on kuvassa värjätty keltaisella. Näistä muutospaikoista keskimmäisen kerroksen muutospaikat ovat keskenään vaihtoehtoisia: 2a, 2b ja 2c.

- Muutoskohta 1 määrittää, miten CAN-kehys tulkitaan ja millä hierarkioilla kehyksen sisältämät tietueet lähetetään viestinvälittäjälle.
- Muutoskohta 2 määrittää loogisen toiminnan, miten viestiä käsitellän: viestin vaatiessa monimutkaista laskentaa tarvitaan uusi komponentti (2a). Jos viestiä ei tarvitse muokata, tai viestin perusteella tehdään yksinkertaista ehtolauseisiin perustuvaa logiikkaa, lisätään tarvittavat konfiguraatiot *MessageForwarder*-moduulin (2b), tai *Algebra*-moduulin (2c) konfiguraatio-tiedostoihin.

- Muutoskohdat 3 ja 4 ovat vain viestin lopullisia vastaanottajia, eivätkä siten lisää monimutkaisuutta viestin etenemisreittiin.
- Muutoskohta 5 määrittää miten uusi viesti lokalisoidaan käyttäjälle.

Muutoskohtia on siis kokonaisuudessaan neljästä viiteen, riippuen siitä esitetäänkö lukua käyttöliittymässä tai tallennetaanko se tietokantaan. Suurin osa vastaanotetuista viesteistä käsitellään XML-konfiguroitavissa *MessageForwarder*-, tai *Algebra*-moduulissa, jolloin koodikanta ei roskaannu lukuisilla vain yhdessä variantissa käytettävillä C++-moduuleilla.



Kuva 15 Uuden CAN viestin lisääminen ohjelmaan ja siitä koituvat mahdolliset lisäykset

MessageForwarder- ja *Algebra*-moduulien tavoin ohjelmistossa on lukuisia moduuleja, joiden toimintaa ohjataan konfiguraatioilla. Tämänkaltaiset moduulit ovat luonteeltaan yleiskäyttöisiä, joista suurin osa on kaikkien tuotevarianttien mukana. Ohjelmisto käsittelee huomattavan määrän erilaisia viestejä, mutta suurin osa viesteistä käsitellään lähes samalla tavalla. Konfigurointi mahdollistaa suhteellisen vaivattoman tavan lisätä uusille viesteille käsittelijät ja siten osaltaan helpottaa tuotteen variointia. Kaikkien tuotteiden yhteisessä koodikannassa on rivimäärällisesti noin kolmen neljäsosan verran C++-koodia ja neljäsosan verran XML-konfiguraatioita. Uusia tuotteita tehtäessä C++-lisäysten määrän voi olettaa olevan huomattavasti pienempi kuin konfiguroinnin määrä.

3.5 Arkkitehtuurin tuomat haasteet

Tärkein, ja siten suurin vaikuttaja arkkitehtuurissa on viestinvälittäjä ja sen käyttö. Arkkitehtuurin joustava ja modulaarinen rakenne sekä dynaaminen viestinvälitys parantavat tuotteen muunneltavuutta. Mutta siinä, missä suunnitteluratkaisuilla on hyvät puolensa, on niillä myös varjopuolensa. Dynaamisten yhteyksien käyttäminen muun muassa heikentää koodin luettavuutta ja konfiguraatioiden määrittäessä dynaamisia yhteyksiä luettavuus heikentyy vielä lisää.

Monet haasteet piilevät hierarkiapohjaisessa viestinvälityksessä, mutta haasteet eivät ole realisoituneet toistaiseksi riskien muodossa. Mikäli ohjelmassa ei käytettäisi viestinvälitystä moduulien kommunikoinnissa, moduulien yhteydet saattaisi olla mahdollista selvittää jo kääntövaiheessa. Tällöin virheellisistä yhteyksistä johtuvat viestien katoamiset tai tahattomat viestiyhteyksien luomiset voisi olla helpommin vältettävissä. Koska viestit identifioidaan hierarkioilla, jotka perustuvat merkkijonoihin, kehittäjän tulee olla hyvin tarkkana sen suhteen, että uusia viestejä luodessa ei käytetä jo olemassa olevia viestihierarkioita. Virhe hierarkiassa tai lähetettävän viestin tietotyypissä voi johtaa ohjelman odottamattomaan toimintaan.

Viestinnän ja viestiyhtyksien dynaaminen luonne mahdollistaa sen, että viestintään voi pesiytyä piileviä virheellisiä viestiyhteyksiä. Viestinvälittäjällä on osoitelista kaikista hierarkioista ja niiden kohdemoduuleista heti käynnistysvaiheiden jälkeen, mutta viestien lähettäjiä ei kirjata missään vaiheessa ylös. On siis mahdollista, että mikä tahansa virheellisesti toimiva moduuli voi spontaanisti lähettää viestin jollain tunnetulla hierarkialla, mutta vastaanottajan kannalta väärällä tietotyypillä. Toisaalta moni moduuli voi lähettää dataa samalla viestihierarkialla, mikä voi johtaa viestin vastaanottajien kannalta erikoisiin toimintatiloihin. Nämä haasteet ovat lisänneet testattavuuden tärkeyttä sekä ohjelman sisäisen viestiliikenteen visualisoinnin tarvetta.

Viestinvälitys toimii moitteettomasti suurimmassa osassa tuotteen vaatimuksista, sillä viestintä on luonteeltaan yksittäisten lukuarvojen tai merkkijonojen lähettämistä. Viestinvälitykseen on lisätty neljä täysin erityyppistä viestiä, jota moduulit voi toisilleen lähettää, mutta välillä ilmenee tarve monimutkaisempien viestien lähettämiseen. Moduuleilla on välillä tarve lähettää usean erityyppisen tietueen viestejä, esimerkiksi jotain tietueita. Viestinvälitys ei kuitenkaan tue muuta kuin tiettyjä viestityyppejä, eikä uusien viestityyppien lisääminen ole mielekästä elleivät ne ole yleiskäyttöisiä. Ainoa olemassaoleva tapa lähettää tietorakenne on jakaa se useampaan yksittäiseen viestiin, mikä monimutkaistaa lähetys- ja vastaanottologiikkaa huomattavasti.

Samojen moduulien käyttö useammassa eri tuotekonfiguraatiossa on yksi arkkitehtuurin päävaatimuksista, mutta tämän toteuttaminen ei ole täysin riskitöntä. Kaikille tuotekonfiguraatioille yhteisiin koodeihin tehdyt päivitykset ja korjaukset välittyvät automaattisesti kaikkien tuotekonfiguraatioiden käännöksiin. Tuotteiden jakaessa samoja moduuleja ylläpidettävän koodin määrä vähenee, mutta samalla koodiin tehtävien muutosten vaikutusalue kasvaa. On riski, että tuotteiden jakamaan moduuliin tehtävät korjaukset ja päivitykset toimivat yhdessä tuotteessa, mutta samalla rikkovat

toisen tuotteen toiminnallisuuden. Vain huolellinen tuotekohtainen manuaali- ja automaatiotestaus voi paljastaa toiminnallisuudessa tapahtuvat epätoivottavat muutokset.

4 ATAM-MENETELMÄN SOVELTAMINEN JA TULOKSET

Tässä luvussa käsitellään ATAM-menetelmän muunnosta, analysointia ja analyysin tuloksia. Ensimmäisenä käsitellään ATAM-menetelmän modifikaation oleellisimpia muutoksia ja perustelut niihin (aliuku 4.1). Muunnelman kohdat käyvät tarkemmin ilmi työn kulusta (aliuku 4.2), mistä ilmenevät myös koetut haasteet ja arvio siitä, miten muunneltu ATAM toimi käytännössä. Aliluvut 4.3 (Laatupuu), 4.4 (Skenaariot), 4.5 (Riskit ja riskiteemat) pitävät sisällänsä niitä tuloksia, joita menetelmästä on tarkoitus saada ulos, mutta huomattavasti referoidussa muodossa. Tämän luvun on tarkoitus antaa kattava kuva siitä, miten menetelmä toimi käytännössä.

4.1 ATAM-menetelmän modifikaatio

ATAM-menetelmä on alkuperäisessä muodossaan kustannussyistä käyttökelpoinen vain suurissa projekteissa, missä menetelmän aiheuttamat suhteelliset lisäkustannukset on saatavissa takaisin analyysin tuloksista seuraavien kustannussäästöjen myötä. Keskisuurissa projekteissa täysimittaisen menetelmän kustannukset saattavat tuottaa projektin kokonaiskustannukseen pieniä parannuksia, mutta menetelmä sellaisenaan ei sovellu pieniin projekteihin. Pienissä projekteissa täysimittaisen ATAM-menetelmän käyttäminen ei ole käytännössä mahdollista vähäisten ihmisresurssien, saati kannattavaa menetelmän tuomien suhteellisesti suurien kustannusten myötä. Jos menetelmän oletetaan tuovan noin 10 prosentin kustannussäästöt projektin koko elinkaaren ajalta, analyysin tulisi olla korkeintaan samanhintainen kuin siitä koituvat säästöt. Ilman pitäviä näyttöjä ei voida olettaa että analyysi tuottaisi tuota kymmentä prosenttia suurempia säästöjä, joten pienempien projektien käyttöön tarvitaan edullisempi versio ATAM:sta. Pohjan säästökohteiden kartoittamiseen voi johtaa seuraavan kaavan mukaan:

$$(a * Osallistuja) * (b * Työvaiheet) = (a * b * Hinta)$$

Jos esimerkiksi a=1/2 ja b=1/2, niin:

$$\frac{1}{2} Osallistujat * \frac{1}{2} Työvaiheet = \frac{1}{4} Hinta$$

Todellisuudessa kustannukset eivät mene näin suoraviivaisesti, mutta tätä kaavaa on käytetty ohjenuorana muunnelmaa kehitettäessä. Jotta menetelmän tuloksien laatu ei heikkene huomattavasti kustannusten alentuessa, tulee säästökohteet valikoida tarkoin. Mikäli tehdään oletus, että kaikki analyysiin osallituvat eivät ole onnistuneen analyysin kannalta yhtä tärkeitä, vähemmän tärkeiden osallistujien karsinta ei välttämättä johda

suuriin eroihin analyysin lopputuloksissa. Työvaihdeiden karsiminen ei ole kuitenkaan yhtä suoraviivaista, sillä työvaiheet rakentuvat edellisten vaiheiden tulosten päälle. Pienemmällä osallistujamäärällä menetelmä on kuitenkin mahdollista suorittaa jouhevammin ja tehokkaammin. Mikäli osallistujilla on jo pohjatietämys ATAM-menetelmästä ja analysoitavasta arkkitehtuurista, perehdyttävät työvaiheet on mahdollista jättää pois.

ATAM-menetelmään on tehty tämän diplomityön puitteissa kahdenlaisia muutoksia: resurssimuutoksia sekä rakenteellisia muutoksia. Resurssimuutoksissa pyrittiin valitsemaan pieni määrä osallistujia, jotka tuntevat arkkitehtuurin ja asiakkaan tärkeimmät liiketoimintanäkökulmat jo ennalta. Tämän mahdollisti se, että kukin osallistuja oli jo pitkän aikaa tehnyt yhteistyötä asiakkaan kanssa kehittäessään analysoitavaa tuotetta. Koska kullakin analyysin osallistujalla oli valmiudet antaa luotettavia ja tarkkoja vastauksia hyvin laajasti sekä arkkitehtuurista että tuotteesta ylipäätään, osallistujien määrä voitiin pitää pienenä.

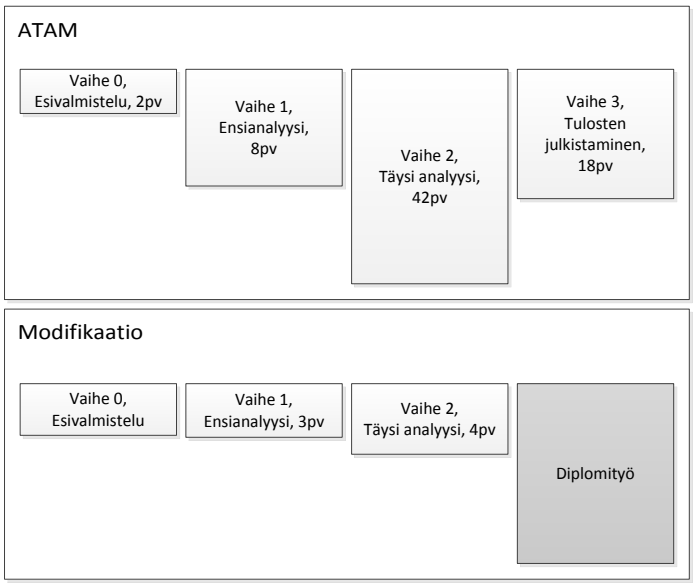
Toisena muutoskohteena oli menetelmän suoritusrakenne. Kun kaikki osallistujat evaluointitiimin johtajaa myöten tuntevat arkkitehtuurin, mitään arkkitehtuurillista perehdytystä ei tarvita. Ainoa jäljelle jäävä esivalmisteluvaihe on ATAM-menetelmään perehdytys, mutta vain siinä määrin, että itse analyysi kulkisi mahdollisimman jouhevasti. Suoritettu modifikaatio oli siis eräänlainen karsittu ATAM.

4.2 Työn kulku

Tässä luvussa käsitellään lyhyesti analyysin kulku alusta loppuun, sekä käydään läpi haasteet, joita työn aikana ilmeni. Työn suorituksen (aliluku 4.2.1) kuvauksessa esitetään analyysin eteneminen käytännön suorituksen kannalta ja itse tulokset käsitellään myöhemmissä aliluvuissa (aliluvut 4.3 - 4.5). Yleisiä haasteita käsittelevä aliluku (4.2.2) täydentää työn suorituksen kuvausta kohdatuilla haasteilla ja lopuksi aliluvussa 4.2.3 esitetään lyhyt arvio analyysin onnistumisesta.

4.2.1 Suoritus

Menetelmä suoritettiin rajallisilla resursseilla niin ajallisesti kuin osallistujamäärällisesti. Aikaa oli varattu kahden kahdeksantuntisen kokouksen verran ja osallistujia evaluoinnin johtajan lisäksi kahdesta kolmeen henkeä (vrt. kuva 16). Jotta evaluointi menisi mahdollisimman jouheasti, varmistettiin, että kukin osallistuja oli tutustunut ATAM-teoriaan jo ennen analyysiä. Perehdytysmateriaalina käytettiin tämän diplomityön teoriaosuutta ja lisämateriaalina teorian lähdeteoksia. Tällä saavutettiin se, että analyysitilanteessa riitti lyhyt teorian kertaus, jotta osallistujat ymmärsivät paremmin suoritettavat työvaiheet.



Kuva 16 ATAM-menetelmän ja sen modifikaation kesto miestyöpäivissä

Ennen ensimmäistä evaluointikierrosta arkkitehtuurista piirrettiin useita erilaisia arkkitehtuurikuvia, joissa arkkitehtuuria tarkasteltiin eri näkökulmista. Arkkitehtuurikuvia piirrettiin myös useista yleisimmistä muutoskohteista, niistä, mihin tuotevariointi eniten vaikuttaa. Koska osallistujat tunsivat arkkitehtuurin ennalta, kuvien merkitys oli lähinnä toimia keskustelunavauksena, muistin virkistämisenä sekä apuvälineenä evaluointitilanteessa. Evaluoinnin edetessä abstrahoidut arkkitehtuurikuvat toimivat hyvänä apuvälineenä arkkitehtuuriratkaisujen tunnistamisessa ja skenaarioiden ideoimisessa. Esivalmisteluissa valmistui myös esitäytetty laatupuu ja esimerkkiskenaarioita, josta analyysin osallistujat saivat paremman kuvan siitä, mitä evaluoinnissa pyritään saavuttamaan.

Ensimmäinen evaluointi tehtiin kolmen henkilön voimin: evaluoinnin johtajan, projektipäällikön ja arkkitehdin kanssa. Ensimmäinen käytännön tilaisuus soveltaa ATAM-menetelmää oli osallistujille oppimistilaisuus, missä myös osoitettiin, että menetelmä on helppo soveltaa käytäntöön. Vaikka teoriasta – menetelmän työvaiheista ja niiden yksityiskohdista – oli hyvät tiedot, ensimmäinen evaluointi jätti paljon parantamisen varaa. Suurimmaksi haasteeksi osoittautui ryhmädynamiikan hallinta siten, että yksilöllisistä näkemyseroista ja toimintatavoista huolimatta pysyttäisiin yhteisessä prosessissa. Prosessista lipsuminen heikensi paikoin menetelmän hyötysuhdetta. Toiseksi suurimmaksi haasteeksi osoittautui teorian eri yksityiskohtien tarkempi tulkinta ja soveltaminen analyysitilanteessa. Osoittautui, että monissa tilanteissa oli tulkinnanvaraisia eroja niin herkkyyskohtien tulkinnan, kuin skenaarioiden vakavuuksien tapauksessa. Muita haasteita on käsitelty myöhemmin aliluvussa 4.2.2. Kohdatuista ongelmista huolimatta ensimmäinen evaluointi tuotti suurimman osan koko analyysin materiaalista.

Toinen analyysi suoritettiin neljän hengen voimin: evaluoinnin johtaja, projektipäällikön, asiakkaan ja sovelluskehittäjän voimin. Resursointisyistä edelliseen evaluointiin osallistunut arkkitehti ei ollut käytettävissä, joten arkkitehdin vastuu siirtyi

evaluoinnin johtajalle. Analyysin anti osoittautui hyvin hedelmälliseksi erityisesti asiakkaan liiketoimintanäkökulmien tarkentumisen ja tuotteen tulevaisuudennäkymien valottumisen kannalta. Edeltävän evaluoinnin suorituksessa tunnistetut ongelmakohdat onnistuttiin suurilta osin välttämään, mikä edesauttoi jälkimmäisen evaluoinnin onnistumista. Toisen vaiheen täysanalyysin myötä skenaarioita laadittiin ja analysoitiin suhteellisen kattavasti arkkitehtuurilta vaadittujen laatuominaisuuksien kannalta. Pienellä ryhmällä oli suhteellisen luonteva edetä prosessissa, mutta osallistujien kokemattomuus ATAM-menetelmän suhteen vaikeutti prosessin kulkua.

Toisen vaiheen täysanalyysissä pureuduttiin asiakkaan näkemyksiin tuotteen tulevaisuudesta ja koska yleisimmät ja todennäköisimmät skenaariot oli jo käsitelty, laaditut skenaariot olivat luonteeltaan tutkivia skenaarioita. Koska itse arkkitehtuuri on kehityksen aikana suoriutunut sille asetetuista vaatimuksista hyvin ja tavallisimmat käyttö- ja muutosskenaariot oli jo analysoitu, jälkimmäisessä analyysissä laaditut skenaariot tarkastelivat harvinaisempia skenaarioita ja puhtaasti hypoteettisia skenaarioita. Hypoteettiset muutosskenaariot käsittelivät toistaiseksi vielä suunnittelemattomia arkkitehtuuriratkaisuja todennäköisten tulevaisuudennäkymien perusteella. Tämänlaisten skenaarioiden laatiminen ei ole ATAM-prosessin mukaista, mutta yleisimpien ja asiakkaan kannalta kiinnostavien skenaarioiden ehdyttyä, skenaarioiden tuoma lisäarvo oli riittävä peruste niiden laatimiseen.

ATAM-menetelmä määrittää myös kolmannen vaiheen, tulosten analyysin. Kukin tähän analyysiin osallistunut henkilö tunsi tuotteen hyvin tarkkaan, eikä riskien meta-analyysin kaltaisen työvaiheen ei nähty olevan tarpeellista. Tähän vaikutti pääosin se, että jälkianalyysistä vastuussa oleva evaluointiryhmä koostui tuotteen kehittäjistä, ei kolmannen osapuolen toimijoista. Tulosten analyysi suoritettiin analyysivaiheiden lomassa, missä sekä kehittäjät että asiakas saivat hyvän kuvan löydetyistä riskeistä, herkkyyskohdista ja tasapainokohdista.

4.2.2 Yleiset haasteet

Teorian soveltaminen käytäntöön ei ollut aivan suoraviivaista. ATAM-teoriassa on ideaalinen näkemys evaluoinnin kulusta, missä käytettävien resurssien määrää ei ole juurikaan rajoitettu ja osallistujilla on selkeät roolit ja riittävä kokemus evaluoinnin läpiajon kannalta. Käytännössä teoriaa vastaavaan ihanneasetelmaan on hyvin vaikea päästä. Haasteita tuovat niin määrärahat kuin osallistujien soveltuvuus evaluointiin. Pienen projektin puitteissa rakennetun pienen evaluointiryhmän jäsenten tulee yhdessä kattaa mahdollisimman laajasti ne roolit, joita ATAM-menetelmässä määritellään. Osallistujien tulee kattaa mahdollisimman hyvin evaluointitiimin, projektin päättäjien ja muiden sidosryhmien roolit ja näkemykset (osallistujat esiteltiin aliluvussa 2.2.1).

Evaluointia järjestettäessä käytettävien resurssien vähyys oli tietoinen haaste, jonka kanssa piti selvitä. Evaluointiin oli mahdollista käyttää diplomityön tekijän lisäksi kolmea muuta arkkitehtuurin kehityksessä mukana ollutta kollegaa sekä asiakkaalta evaluoitavasta tuotteesta vastaavaa henkilöä. Resurssien vähyys itsessään rajoitti evaluointitiimin koon yhteen, sillä diplomityön tekijä oli ainoa käytettävissä oleva

ennestään ATAM-menetelmään tutustunut henkilö. Muilla osallistujilla pyrittiin kattamaan tasaisesti projektin päättäjien ja muiden sidosryhmien näkemykset tuotteesta ja sen arkkitehtuurista. Sekä käytettävien tuotteen tuntevien henkilöiden vähyyden että potentiaalisten osallistujien ajankäytöllisten seikkojen myötä ensianalyysi rajoittui kokonaisuudessaan kolmeen henkilöön ja toinen analyysi neljään henkilöön.

Ensikertalaisuus aiheutti selkeitä haasteita evaluointiin ja ATAM-menetelmän noudattamiseen. Kukin osallistuja oli perehtynyt teoriaan tämän diplomityön teorialuvun muodossa (aliluku 2), mutta kenelläkään, evaluoinnin johtaja mukaan lukien, ei ollut yhtään käytännön kokemusta arkkitehtuurien evaluoinnista. Kokemattomuus näkyi paljon siinä, että osallistujilla oli taipumus ajautua prosessin ulkopuolelle tai käsitellä asioita prosessinvastaisesti. Huomattavaa haastetta lisäsi myös se, että analyysin ohjaamisen lisäksi evaluoinnin johtajan piti samanaikaisesti toimia prosessiinpakottajan roolissa ja suorittaa tarkkaa arkkitehtuurianalyysiä. Analyysitilanteen tarkastelu jälkikäteen johtaa siihen tulokseen, että aikaisempi kokemus menetelmästä on ensiarvoisen tärkeää, koska muutoin analyysissä nojaudutaan liikaa yksilölliseen kykyyn onnistua monimutkaisesta toimeksiannosta – ensimmäisellä yrityksellä.

Toiseksi suureksi haasteeksi osoittautui piirteiden kategoroisointi *herkkyyskohdiksi*, *tasapainokohdiksi* tai *riskeiksi*. Ongelmaksi osoittautui se, että piirteen määrittely riskiksi tai herkkyyskohdaksi ei ollut täysin yksiselitteistä. Alkuvaiheessa kaikki löydökset vaikuttivat olevan riskejä, vaikka asia ei näin ollutkaan. Ratkaistavaksi haasteeksi tuli määritellä se häilyvä raja, milloin piirre lakkaa olemasta riski ja onkin vain herkkyyskohta. Myös pientä epäselvyyttä aiheutti herkkyyskohtien mahdolliset sekundääriset vaikutukset ja se milloin niiden vaikutus on riittävän merkittävä muuttaakseen herkkyyskohdan tasapainokohdaksi.

4.2.3 Arvio onnistumisesta

Analyysin suoritus oli jokaiselle osallistujalle ensimmäinen käytännön kokemus ATAM-menetelmästä. Tämä kokemattomuus paistoi ensimmäisessä evaluoinnissa, mistä johtuen ATAM-menetelmän noudattaminen koitui kohtalaisen haasteelliseksi. Pelkästään arkkitehtuurin riskien tunnistaminen, aikataulussa pysyminen ja evaluoinnin rakenteen ylläpitäminen vaativat evaluoinnin johtajalta käytännön kokemusta. Tästä johtuen analyysin tavoitteisiin ja odotuksiin suhtauduttiin maltillisesti ja koko prosessin tavoitteena oli saavuttaa mahdollisimman paljon olemassa olevilla resursseilla ja kokemuksella.

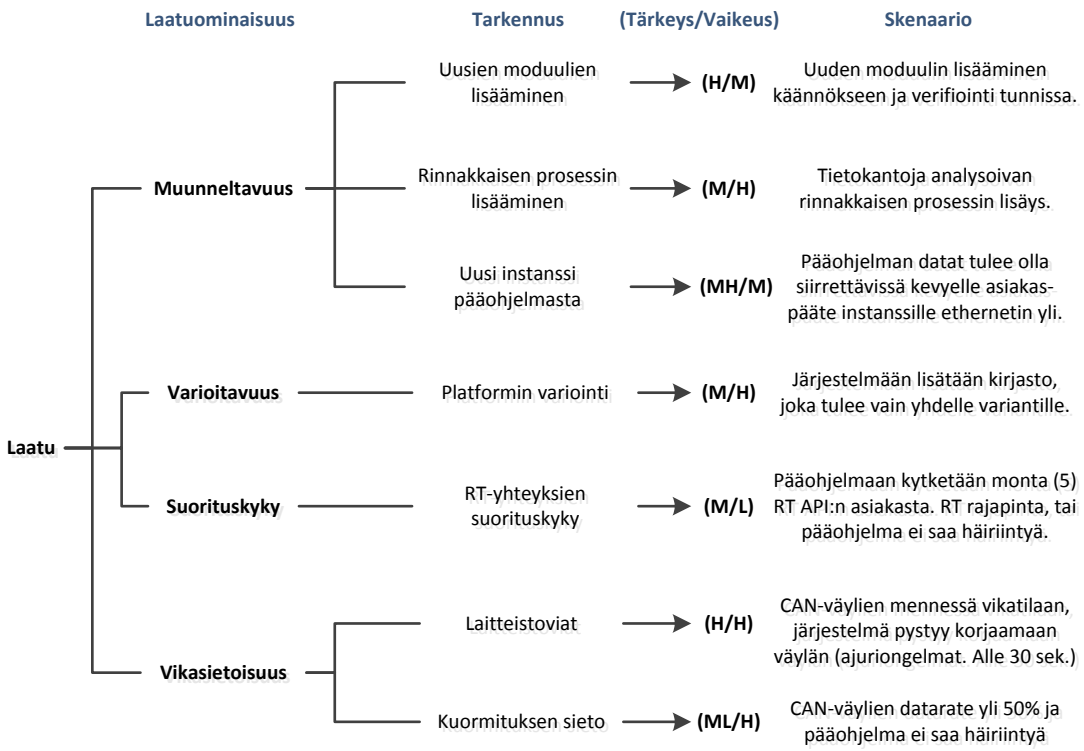
Evaluoinnissa ensimmäinen tehtävä on arvioida olemassa olevan dokumentaation riittävyys, sillä puutteellisilla tiedoilla evaluointia ei voi suorittaa. Koska evaluoijat tunsivat arkkitehtuurin jo entuudestaan, oli ilmeistä, että evaluoinnin olisi voinut suorittaa jopa ilman mitään dokumentaatiota. Ohjelmistosta ja arkkitehtuurista oli olemassa hyvin kattava dokumentaatio, jonka tarve osoittautui tämän evaluoinnin tapauksessa hyvin pieneksi.

Koska dokumentoitu arkkitehtuuri oli evaluoinnin aikana jo hyvin kypsällä asteella, haasteeksi tuli saada evaluoinnista jotain hyötyjä. Suurin osa arkkitehtuurin ongelmista oli korjattu kehitysvaiheessa, eikä kehittäjillä ollut tiedossa mitään vakavaa riskiä, mikä voisi kaataa koko arkkitehtuurin. Evaluoinnissa kävi kuitenkin niin, että uusia riskejä löydettiin, mutta mitään erityisen vakavia riskejä ei löydetty. Uusien riskien löytymisen voi perustella johtuneen siitä, että ohjelmiston tehtävät ja vaatimukset olivat laajentuneet siitä, kun arkkitehtuurista ja moduuleista oli tehty ensimmäinen tuoteiteraatio. Kun kolmannen toteutetun tuotteen jälkeen arvioitiin moduuleja, jotka oli tehty ensimmäiseen tuotteen aikana, jotkin riskit olivat lähes ilmeisiä.

Vaikka ensimmäinen evaluaatio oli haastava, jälkimmäisen analyysin loppuun mennessä oli saatu aikaiseksi kattava laatupuu ja nippu skenaarioita. Arkkitehtuurista onnistuttiin tunnistamaan riskejä, herkkyyskohtia ja tasapainokohtia. Aika näyttää kuinka hyödyllisiksi nämä löydökset osoittautuvat, mutta evaluoinnin kenties paras anti ilmeni tiedonjakamisen muodossa asiakkaan ja projektiryhmän välillä. Asiakkaan ollessa teknisessä vastuussa tuotteesta tuotteen rakenne ja rajoitteet antoivat asiakkaalle erittäin hyvää tietoa siitä, miten tuotetta voi ja kannattaa jatkossa kehittää. Lisäksi tutkivia ja hypoteettisia skenaarioita laadittaessa projektiryhmä sai asiakkaalta hyödyllistä tietoa siitä, minkälaisia tulevaisuudennäkymiä tuotteella on. Kokonaisuudessaan voidaan arvoida, että alkuperäiset tavoitteet saavutettiin ja evaluoinnista oli projektille hyötyä.

4.3 Laatupuu

Laatupuuhun laadittiin kokonaisuudessa 26 skenaarioita viidelle laatuominaisuudelle, jotka ovat muunneltavuus, varioitavuus, testattavuus, suorituskyky ja vikasietoisuus. Jokainen laatuomainaisuus on perusteltavissa sekä asiakkaan liiketoimintanäkökulmien että toiminnallisten vaatimusten perusteella. Aikaisemmissa luvuissa on käynyt ilmi, että muunneltavuus ja varioitavuus ovat tärkeimmät ominaisuudet, jotta tuotekehityskustannukset pysyisivät hallinnassa. Kehitysaikaisen testattavuuden tärkeys korostuu siinä, että loppuasiakkaan hyväksyntätestaus on aina ollut erittäin tarkka. Vikasietoisuus on perusteltavissa sillä, että vaatimusten mukaan ohjelmiston huoltotarpeen tulee olla erittäin pieni. Suorituskyky on tärkeä sekä ohjelman reaaliaikavaatimusten että mahdollisten myöhäisempien varianttien laajemman toiminnallisuuskirjon takia. Nämä laatuominaisuudet on nähtävissä alla olevasta laatupuusta (kuva 17), johon on valittu vain suppea osuus koko laatupuusta.



Kuva 17 Typistetty otos laatupuusta

Laatupuun laadinnassa skenaarion tärkeys ja toteuttamisen vaikeus on skaalattu viisiportaiseksi: L (Low), ML (Medium-Low), M (Medium), MH (Medium-High) ja H (High). Suurimmassa osassa skenaarioista kolmiportainen skaala (L, M, H) oli riittävä, mutta muutamassa tapauksessa skenaarion tärkeydestä tai vaikeudesta ei päästy yksimielisyyteen. Tällöin turvauduttiin väliasteisiin (MH ja ML), jolloin skenaarioiden arvottamisesta päästiin yhteisymmärrykseen ja lopullinen analysoitavien skenaarioiden lista saatiin muodostettua. Esimerkkinä tärkeähkö skenaario (MH) saattoi olla sellainen, joka kohtuullisen suurella todennäköisyydellä toteutuu tulevaisuudessa, mutta on odotettavissa, että toteutunut skenaario ei ole täysin analysoidun skenaarion tapainen. Vakavuuksien väliasteet eivät tuo analyysiin erityisen paljon lisäarvoa, mutta niillä purettiin väittelytilanteet nopeasti kaikkien hyväksymänä kompromissiratkaisuna.

4.4 Skenaariot

Tässä aliluvussa esitellään laadituista ja analysoiduista skenaarioista kaksi, joissa tutkitaan arkkihtuuria eri näkökulmista. Ensimmäinen skenaario käsittelee todennäköistä asiakasvaatimusta ja osuu tutkittavista laatuominaisuuksista muunneltavuuden kategoriaan. Toinen skenaario on luonteeltaan hypoteettinen. Skenaariossa tutkitaan epätodennäköistä vikatilannetta ja sitä, miten arkkitehtuuri selviää siitä. Skenaarioita analysoitiin yhteensä seitsemän kappaletta, joista valikoidut kaksi osoittautuivat kiinnostavimmiksi.

Skenaariot esitellään kolmessa osassa. Ensin skenaariot esitellään lyhyesti ja selväsanaisesti. Tämän jälkeen esitellään ne osat arkkitehtuurista, jotka ovat olennaisia

analyysin kannalta. Tässä esitellään skenaarioon liittyviä arkkitehtuuriratkaisuja tarkemmin kuin mitä aikaisemmin aliluvussa 3.3 on esitelty. Lopuksi esitellään skenaarion analyysi ja siitä löydetyt herkkyys- ja tasapainokohdat, riskit ja ei-riskit. Skenaarion analyysi esitellään sekä taulukkojen avulla että selväsanaisesti.

4.4.1 Skenaario 1 – Käyttöliittymän datojen peilaus

Ensimmäinen skenaario käsittelee tapausta, missä pääohjelman käyttöliittymän datat peilataan etäkäyttöliittymälle. Tämä skenaario on luonteeltaan tutkiva skenaario, joka ei pohjaudu ohjelmiston vaatimuksiin. Skenaario on laadittu asiakkaan liiketoimintanäkökulmien perusteella todennäköisenä tulevaisuuden vaatimuksena. Koska skenaarion toiminnallisuus ei ole vaatimuksissa, siitä ei ole myöskään virallisesti julkaistua toteutusta. Analyysissä käytettävä toteutus on prototyyppi, minkä toimintaa on testattu kehitysympäristössä. Tässä aliluvussa käydään läpi tarkempi esittely, prototyypin arkkitehtuurikuvaus ja itse analyysi.

Skenaarion esittely

Skenaariossa tutkitaan tilannetta, jossa samaa reaaliaikadataa tarkastellaan kahdelta eri tietokoneisiin liitetyiltä päätteiltä samanaikaisesti. Molemmat käyttöliittymät ovat sillä tavoin itsenäisiä, että käyttöliittymäinteraktiot eivät näy päätteeltä toiselle. Tähän skenaarioon liittyvästä arkkitehtuuriratkaisusta on tehty prototyyppitoteutus pienemmässä mittakaavassa ja skenaariossa on tarkoitus tutkia toteutuksen soveltuvuutta loppukäytäjän käyttöympäristössä.

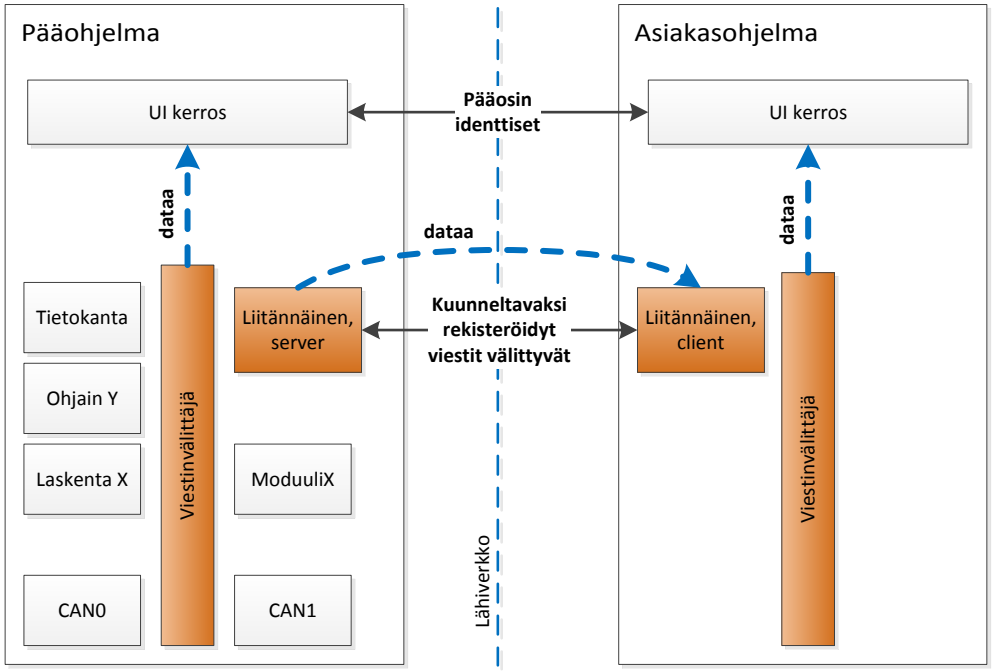
Skenaario laadittiin ensimmäisessä evaluoinnissa ennakoiden asiakkaan tarpeita tunnettujen liiketoimintanäkökulmien perusteella. Loppukäyttäjän järjestelmässä on jo olemassa laite, joka toistaa pääohjelman tuottamaa dataa, mutta kyseinen ohjelmisto tehdään eri lähdekoodeista ja käyttää datalähteenään reaaliaikarajapintaa. Tässä skenaariossa tutkitaan arkkitehtuuriratkaisua, joka mahdollistaa sen, että pää- ja asiakasohjelma voivat käyttää yhteistä ydinarkkitehtuuria ja ne voidaan kääntää samoista lähdekoodeista pienellä erolla moduulikokoonpanossa. Tässä skenaariossa on tehty oletus, että asiakasohjelmalla ei ole muuta tehtävää kuin pääohjelman datan toistaminen.

Skenaarioon liittyvät arkkitehtuuriratkaisut

Skenaarion keskeisin arkkitehtuuriratkaisu perustuu konseptiin, missä pääohjelma ja asiakasohjelma jakavat saman viestinvälittäjän. Ohjelmiston kaikki moduulit kommunikoivat viestinvälittäjän välityksellä, joka ei välitä mistä viesti on alkujaan viestijonoon lisätty. Tämä lähettäjästä riippumattomuus säilyy siinäkin tapauksessa, jos pääohjelman viestijonon viestit välitetään jonkin samaa arkkitehtuuria käyttävän asiakasohjelman viestijonoon. Koska kaikki käyttöliittymällä esitettävä data vastaanotetaan viestinvälittäjältä samalla mekanismilla, jolla moduulit kommunikoivat, käyttöliittymän kannalta ei ole eroa onko, data tuotettu samassa vai eri ohjelmassa.

Tällöin samoista lähdekoodeista käännetyn asiakasohjelman, jolla ei ole dataa tuottavia moduuleja, pitäisi pystyä toistamaan pääohjelman tuottamat datat identtisesti.

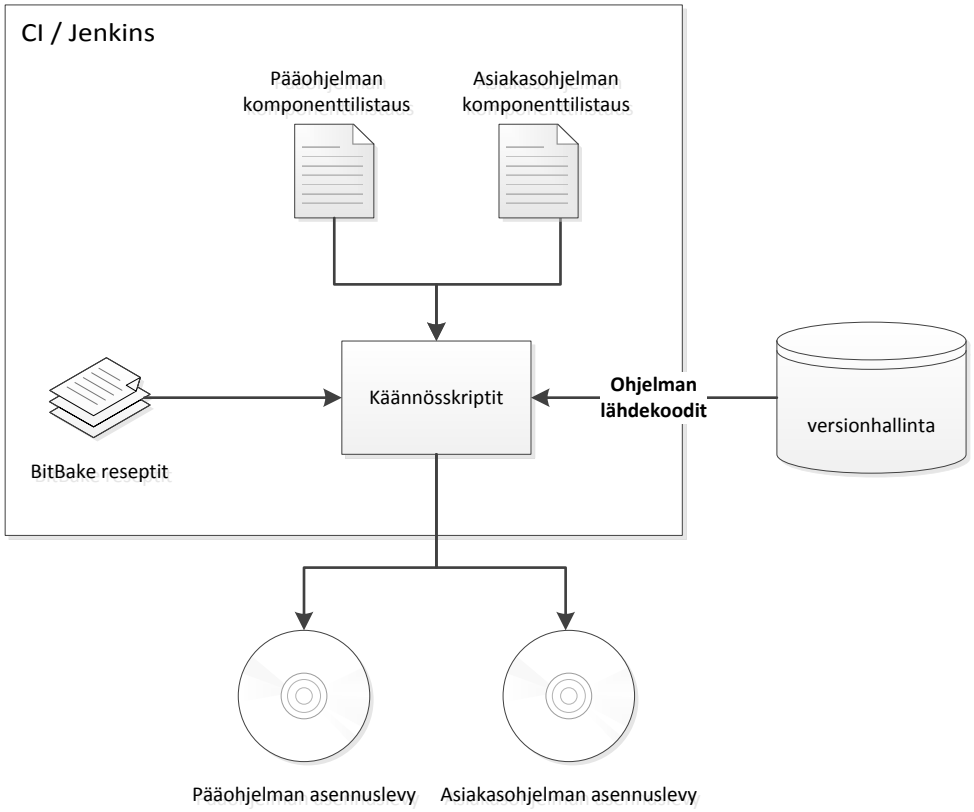
Viestinvälittäjien yhdistämiseksi on tehty prototyyppinä liitännäiset, jotka pureutuvat suoraan viestinvälityksen ydinkoodiin. Liitännäisistä on sekä palvelin- että asiakasversio, jolloin yhteydenmuodostuksen vastuu jää asiakasohjelmalle. Kumpikaan liitännäinen ei muuta viestinvälittäjien toimintalogiikkaa ohjelmien peruskäytön tapauksessa, vaan liitännäiset aktivoimalla tulee vain ylimääräinen kommunikaatioväylä kahden ohjelman välille. Tämän skenaarion tapauksessa asiakasliitännäinen kommunikoi palvelinliitännäiselle haluamien viestihierarkioiden rekisteröimisen (rekisteröityminen esitetty aikaisemmin aliluvussa 0) ja palvelin lähettää asiakkaalle sen vaatimat viestit. Jotta turha dataliikenne ei aiheuttaisi suorituskykyongelmia, pääohjelma välittää viestejä vain niillä viestihierarkioilla, joilla asiakasohjelman moduulit ovat rekisteröineet. Yhteydet ovat TCP-yhteyksiä, jolloin kommunikointi toimii luotettavasti. Kuva 18 esittää, miten kahden ohjelman viestinvälittäjät synkronoituvat.



Kuva 18 Arkkitehtuurikuva viestinvälittäjien synkronoinnista

Liitännäiset mahdollistavat myös molemminsuuntaisen viestinvälittäjien synkronoinnin, jolloin asiakasohjelma voisi tuottaa dataa pääohjelmalle. Tämä voisi johtaa siihen, että asiakasohjelma saattaisi joissain tapauksissa häiritä pääohjelman toimintaa. Tämän ja muiden selkeiden riskitekijöiden myötä analyysissä ei käyty läpi prototyyppiratkaisua, vaan sen jatkokehitysratkaisua. Prototyyppiin oli tehty kaksi muutosta, joista ensimmäisenä muutoksena päätettiin datan välityksen olevan vain palvelinliitännäiseltä asiakasliitännäiselle päin. Toisena muutoksena huomioitiin kommunikoinnin aiheuttamat potentiaaliset suorituskykyongelmat, mistä johtuen päätettiin palvelinliitännäisen suodattavan lähettämäänsä dataa maksimissaan kymmenen muutoksen sekuntitahtiin.

Myöhemmin skenaarion analyysissä tulee ilmi käännösjärjestelmässä piilevä ongelma, joten myös käännösjärjestelmän kuvaus on syytä käydä läpi. Kuva 19 esittää käännösjärjestelmän yksinkertaistetusti niiltä osin, mihin ongelma liittyy. Kaikki variantit, kuten tämän skenaarion kaksi ohjelmaa, tehdään samoilla käännösskripteillä. Kaikkien varianttien koodit ovat versionhallinnassa niputettu samaan paikkaan, josta skriptit suodattavat käännökseen sekä ydinarkkitehtuurin koodit että tuotekohtaista komponenttilistausta vastaavat koodit. Samassa käännöksessä käännetään pääohjelman lisäksi myös kohdelaitteeseen asennettava Linux-käyttöjärjestelmä kaikkien ohjelmariippuvuuksien kanssa. Lopullinen käännös, jolloin käännetään pääohjelma sekä käyttöjärjestelmä, tapahtuu BitBake-käännösreseptien määritelmien mukaan. BitBake on käännösreseptejä käyttävä käännösjärjestelmä, millä voi kääntää samanaikaisesti sekä sovelluksen että asennettavan Linux-käyttöjärjestelmän, johon sovellus asennetaan.



Kuva 19 Yksinkertaistettu kuvaus käännösjärjestelmästä

Skenaariossa ilmi käyvä ongelma liittyy juuri BitBake-käännösresepteihin. Reseptejä päivittämällä voidaan muunnella Linux-alustan kokoonpanoa esimerkiksi niiltä osin, mitä ohjelmia ja ajureita sinne asennetaan. Ongelma on siinä, että nämä reseptit eivät ole varioitavia. Kaikki reseptimuutokset välittyvät suoraan kaikille uusille ja edeltäville tuotteille. Tästä johtuen käännösreseptit rajoittavat kankeasti sen, mitä tuleville tuotevarianteille voi tehdä, sillä uusia variantteja tehdessä ei saa tulla sellaisia alustamuutoksia, jotka rikkovat vanhojen tuotteiden toiminnallisuuden. Tämän skenaarion puitteissa ongelma tulee Linux-alustan resepteissä määritellystä staattisesta

IP-osoitteesta, joka asettaa palvelintietokoneen ja asiakastietokoneen IP-osoitteet samoiksi.

Skenaarion analysointi

Skenaarion analyysi on eritelty kahteen taulukkoon: analyysitaulukkoon (taulukko 3) ja löydöksien tarkempien yksityiskohtien taulukkoon (taulukko 4). Skenaariossa on käytetty muutamia termejä, jotka ovat arkkitehtuurispesifejä. Tässä skenaariossa käytetään viestinvälittäjän liitännäisestä lyhennettä RMI (Remote Messaging Interface, etäviestintärajapinta). RMI-server on pääohjelman liitännäinen ja RMI-client on asiakasohjelman liitännäinen. Taulukoissa on käytetty seuraavia lyhenteitä: riski (Rx), turvallinen ratkaisu, eli ei-riski (NRx), herkkyyskohta (Sx) ja tasapainokohta (Tx).

Taulukko 3 Ensimmäinen skenaario: Käyttöliittymän datojen peilaus

Skenaario #	Isäntäkoneen tuottama käyttöliittymädata tulee saada siirrettyä myös toisen tuotevariantin käyttöliittymälle ethernetin yli.		
Laatuominaisuus	Muunneltavuus		
Ympäristö	Normaali toiminta		
Ärsyke	Isäntäkone lähettää dataa toisella tietokoneella olevan asiakasohjelman käyttöliittymälle.		
Vaste	Asiakasohjelma näyttää samaa dataa kuin isäntäohjelma		
Suunnitteluratkaisu	**Riski / Ei-Riski**	**Herkkyyskohta**	**Tasapainokohta**
RMI-server		S1, S2	
RMI-serverin viestien välimuisti			T1
RMI-server voi vain lähettää dataa	NR1		
RMI-server suodattaa viestit 100 ms intervallille	NR2	S3	T2
RMI-client voi vain vastaanottaa dataa	R1, NR3		
Qt-ohjelmistokehys	R2		
Kaikki laskentalogiikka vain pääohjelmalla	NR4		
BitBake-reseptit	R3		
Lähiverkko	R1	S4	
Järkeily	Arkkitehtuurikuvat on esitetty aikaisemmin tässä luvussa.		

Taulukko 4 Ensimmäisen skenaarion analyysin tarkennukset

Kohta	Järkeily
R1	Verkkokatkoksen aikana ruudulle jää vanhat tiedot. Vanha tieto on virheellistä dataa, eikä nykyisellä mekanismilla voida indikoida tätä.
R2	Qt:n verkkoteutuksessa on ollut virheellistä toimintaa, mikä on aiheuttanut kaatumisia käytettäessä QtcpSocket-luokkaa.
R3	BitBake-reseptit eivät ole varioitavissa ja molempiin Linux-alustoihin tulee samat IP-osoitteet. IP-osoitteet ovat staattisia, eikä lähiverkossa ole DHCP-palvelinta.
S1	Pitää olla nopea verkko, jotta RMI-toteutus on mahdollista.
S2	RMI:n lisäys lisää pääohjelman kuormitusta. Mikäli kuormitus kasvaa liian suureksi, voi koitua suorituskykyongelmia.
S3	Suodatus parantaa pääohjelman suorituskykyä.
S4	Verkkoviiveet ja lähiverkon kuormitusaste voi näkyä vasteajoissa.
T1	Välimuisti ylläpitää tietoa kaikista viesteistä, jotta myöhemmin liitetyt asiakkaat saavat kaikki datat. Pieni heikentävä vaikutus suorituskykyyn, joka riippuu mm. puskuroitavien viestien määrästä. (luotettavuus+, suorituskyky-)
T2	Parantaa asiakasohjelman suorituskykyä, mutta kasvattaa latenssia.
NR1	Jos arkkitehtuurissa oletetaan, että pääohjelma ei tarvitse asiakasohjelman tuottamaa dataa, tällä varmistetaan että asiakasohjelma ei häiritse pääohjelman loogista toimintaa.
NR2	Suodatuksessa ei ole ongelmia niin kauan kun vastaanottaja vain näyttää tietoja. Mikäli datan perusteella tehtäisiin laskentaa, suodatus saattaa johtaa loogisesti virheelliseen toimintaan.
NR3	Ei riskiä pääohjelman loogiselle toiminnalle.
NR4	Yhden ohjelmalogiikkakerroksen pitäminen parantaa muunneltavuutta: On helpompi lisätä pelkkä käyttöliittymäkerros asiakasohjelmaan.

Skenaariosta löydettiin kolme riskiä, joista yksi (R3) oli vakava ja sellaisenaan estää skenaariota vastaavan ominaisuuden toteuttamisen. Vaikka kyse onkin vain yhdestä IP:stä, ongelman korjaaminen vaatii käännösjärjestelmään BitBake-reseptien varioimisen tukea tai ajoaikaista IP-konfigurointien muokkaamista asiakassovelluksen ensikäynnistyksellä. Riski R2 perustuu käytännön tietoon siitä, että Qt:n versiossa 4.7.x TCP-socketin toteutus on ollut virheellinen ja vaikka Qt:n uudemman version 4.8.x verkkototeutus on osoittautunut luotettavammaksi, siihen suhtaudutaan varauksella. Verkkototeutuksen virhe voi aiheuttaa ohjelman kaatumisen yhteydenmuodostuksessa, mikä on vakava ongelma luotettavuuden kannalta. Lievin riski R1 on vain siinä prototyyppitoteutuksessa, jota analyysissä tutkittiin. Riski on korjattavissa yksinkertaisella muutoksella RMI-client moduuliin puskuroimalla kaikkea sisäänmenevää dataa ja nollaamalla niiden arvot yhteyden katketessa.

Skenaariossa oli lukuisia herkkyys- ja tasapainokohtia, joissa huomioitiin yhteys ohjelman suorituskykyyn. Järjestelmässä kulkee viestejä suurella nopeudella ja tulevaisuuden varianteissa viestitulva voi olla suurempi. Viestien lähetystahti vaikuttaa suorituskykyyn jokaisessa paikassa, missä niitä käsitellään, joten suodatus on nähty oleelliseksi toimenpiteeksi (S2, S3). Suodatus parantaa asiakassovelluksen suorituskykyä, mutta lisää latenssia viestipäivitysten välittämisessä asiakassovellukselle. Myös verkon suorituskyky on huomioitu, sillä niin hidas verkkoyhteys kuin kuormitettu lähiverkko voivat aiheuttaa ongelmia viestien lähettämisessä asiakassovellukselle (S1, S4).

Verkon yli käytävässä viestiliikenteessä on huomioitu, että asiakassovellus ei välttämättä ollut muodostanut yhteyttä ennen kuin ensimmäiset viestit olivat lähteneet. Tästä johtuen kaikkia lähetettäviä viestejä on puskuroitu, jotta asiakassovellus saa heti yhteydenmuodostuksessa kaikista viestityypeistä viimeisimmät statukset (T1). Tämän toiminnallisuuden avulla voidaan luottaa, että kaikki tarvittava data välittyy asiakassovellukselle riippumatta siitä, milloin sen kytketään, mutta palvelinsovelluksen kuormituksen hinnalla.

Tämän skenaarion tapauksessa löytyi yksi riski (R3), joka on niin vakava, että riskin kiertäminen vaatii huomattavia toimenpiteitä. Skeenaarion esittämää toiminnallisuutta ei ole mahdollista toteuttaa sellaisenaan, mutta positiivisena seikkana on huomioitava, että tämä toiminnallisuus ei ole ohjelmiston vaatimuksissa. Muut riskit eivät olleet vakavia, eikä herkkyys- tai tasapainokohdissa ollut havaittavissa mitään huolestuttavia piirteitä. Skenaariosta voi päätellä, että jos jotain tämänkaltaista toiminnallisuutta halutaan, se kannattanee toteuttaa mahdollisuuksien mukaan toimivaksi todetulla RT-rajapinnalla.

4.4.2 Skenaario 2 – Viestitulva CAN-väylästä

Tässä skenaariossa tutkitaan arkkitehtuurin ominaisuuksia poikkeuksellisessa tilanteessa, jossa CAN-väylien viestimäärä kasvaa huomattavan suureksi. Tämä skenaario on tutkiva siinä mielessä, että tilanne ei ole erityisen todennäköinen, mutta tämä saattaa tuoda ilmi arkkitehtuurissa piileviä riskejä, jotka voivat ilmetä muissa yhteyksissä. Tämä on esimerkkiskenaariona hyvä, sillä lähtöasetelma ja analyysin tulokset ovat muita skenaariota selkeämpiä ja helpommin ymmärrettäviä. Seuraavissa aliluvuissa käydään tarkemmin läpi skenaarion esittely, skenaarioon liittyvät arkkitehtuuriratkaisut ja analyysin tulokset.

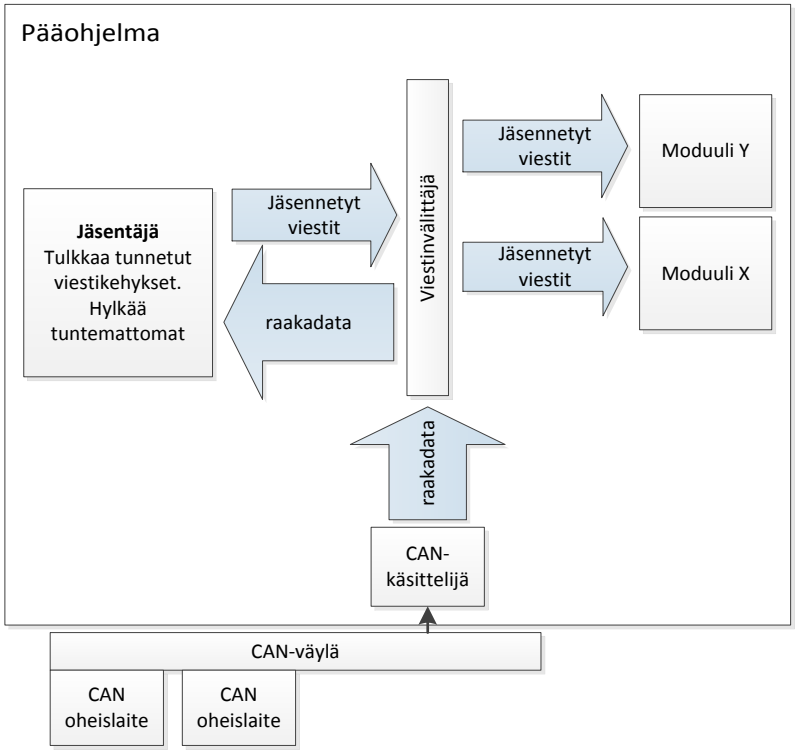
Skenaarion esittely

Skenaariossa käsitellään tilannetta, jossa CAN-oheislaite lähettää dataa huomattavasti normaalia nopeampaan tahtiin joko vikaantumisen tai jonkin muun syyn takia. Pääohjelma vastaanottaa ja käsittelee kaikki CAN-viestit, mistä johtuen suuri datamäärä voi aiheuttaa suorituskykyongelmia. Käytettävissä olevien prosessoritehojen lisäksi suuri vaikutus suorituskykyyn aiheutuu arkkitehtuuriratkaisuista. Tässä skenaariossa keskitytään kartoittamaan mahdolliset arkkitehtuurilliset pullonkaulat ja riskitekijät.

Käytännön suorituskykymittauksista saatujen tulosten perusteella tavallinen viestimäärä vastaa väylästä riippuen 3 – 10 % CAN väylän maksimi-kuormituksesta. Mittausten perusteella jokaisen CAN-väylän viestimäärä pysyy suhteellisen tasaisena koko ohjelman suorituksen ajan, eikä pääohjelman tuottama kommunikaatio lisää väylän kuormitusastetta juurikaan. Mikäli yhdeltä väylältä lähetetään viestejä puolella sen maksimaalisesta suorituskyvystä, väyläkohtainen kokonaisviestimäärä 5 – 17 kertaistuu. Tavallisen suorituskykyä tutkivan skenaarion luonteeseen kuuluu, että esimerkiksi datamäärä kymmenkertaistetaan, mutta on mahdollista, että tulevaisuuden tuotevarianttien tapauksessa tämä lukema saattaa vastata tavallista käyttötilannetta.

Skenaarioon liittyvät arkkitehtuuriratkaisut

Vastaanotettujen CAN-viestien tietosisällöt etenevät arkkitehtuurin sisällä hyvin johdonmukaisesti tiettyjen yhteisien vaiheiden läpi (kuva 20). Vastaanotetut CAN-kehykset jakautuvat joko yksittäisiin viesteihin tai jotain monikehysprotokollaa (esim. KWP2000) noudattaviin viesteihin. Yksittäisistä kehyksistä koostuvat viestit ovat sellaisenaan tulkattavia ja ne lähetetään suoraan jäsentäjämoduulille. Jäsentimen tehtävä on tulkata tunnetuista viesteistä niiden tietosisältö ja hylätä tuntemattomat viestit. Jäsennetyt viestit lähetetään edelleen viestinvälittäjälle, joka välittää viestit niiden kohdemoduuleille.



Kuva 20 CAN viestien tietosisällön eteneminen pääohjelmassa

Viestien vastaanotto- ja jäsennysmekanismi pysyy muuttumattomana koko ohjelman suorituksen ajan. Tietyillä tunnisteilla varustetut CAN-viestit aiheuttavat aina suurin piirtein samanlaisen vasteen ja siten ohjelman toiminta kunkin tunnisteen tapauksessa on hyvin ennustettavissa. Jäsentäjän jälkeinen toiminnallisuus tapahtuu yksittäisissä moduuleissa ja aktivoituvat moduulit ovat riippuvaisia suoraan siitä, mitä CAN-viestejä

ohjelmisto on vastaanottanut. Vaikka mahdolliset suorituskykyongelmat voivat ilmentyä juuri näistä jäsennettyjä viestejä käsittelevistä moduuleista, suurin osa niistä jätetään tämän analyysin piirin ulkopuolelle. Useista kymmenistä eri moduuleista käsitellään tämän skenaarion puitteissa vain perustoiminnallisuuden kannalta oleellisimmat moduulit.

Tämän skenaarion tapauksessa kuormituksen oletetaan tulevan CAN-viesteistä, mutta viestien vaikutus pääohjelman kuormitukseen vaihtelee huomattavasti viestien välillä. Kaikkein vähiten kuormittava CAN-viesti on sellainen, jossa ei ole pääohjelman kannalta kiinnostavaa informaatiota ja viesti hylätään jäsennysvaiheessa. Kiinnostavaa tietoa sisältävät viestit kuormittavat järjestelmää moninkertaisesti enemmän, sillä ne etenevät arkkitehtuurin sisällä useimmiten käyttöliittymälle ja rajapinnoille asti. Koska CAN-viesti voivat sisältää useita eri tietueita, viestikohtaiset kuormituserot voivat olla yli kymmenkertaisia. Myös tietueiden arvoilla voi olla vaikutusta kuormitukseen, sillä moduulien logiikka voi toimia sekä arvojen muutosten että tiettyjen lukuarvojen tapauksessa eri tavalla.

Skenaarion analysointi

Siitä huolimatta, että skenaarion ärsykkeeksi määriteltiin tarkka kuormitustaso, analyysiä ei suoritettu tarkkojen laskelmien avulla. Analyysissä keskityttiin mahdollisiin pullonkauloihin, ja koska laskelmien tekeminen olisi voinut osoittautua haasteelliseksi, analyysi eteni kokemusperäisten arvioiden perusteella. Loppujen lopuksi skenaarion esittämä datamäärä on huomattavasti suurempi kuin mihin ohjelma pystyy nykyisillä arkkitehtuuriratkaisuilla vastaamaan. Riskit olivat osittain tiedossa jo ennen skenaarion analyysiä, mutta ATAM-menetelmä antoi odotusten mukaisesti myös uutta tietoa. Taulukot 5 ja 6 sisältävät analyysin tulokset, joissa on esitetty kaksi riskiä ja usea muu positiivinen ja negatiivinen ominaisuus. Siitä huolimatta, että tämän skenaarion riskeissä ja herkkyyskohdissa on käytetty samoja merkintöjä (R1, R2, S1, T1...) kuin edellisessä skenaariossa, niillä ei ole mitään yhteistä keskenään.

Taulukko 5 Toinen skenaario: Suuren CAN-viestimäärän käsittely

Skenaario #	Oheislaite vikaantuu ja jokaisen CAN-väylän kuormitus kasvaa yli 50-prosenttiin väylien maksimikuormasta. Pääohjelman toiminta ei saa häiriintyä.		
Laatuominaisuus	Vikasietoisuus, suorituskyky		
Ympäristö	Oheislaite vikaantunut ja dataa tulee paljon		
Ärsyke	Dataa syötetään yli 50 % CAN-väylien maksimikuormasta		
Vaste	Ohjelman toiminta ei häiriinny		
Suunnitteluratkaisu	**Riski / Ei-Riski**	**Herkkyyskohta**	**Tasapainokohta**
CAN kirjasto/ajuri	NR1, R1		
Kuunneltavaa dataa ei suodateta	R2		T1
Kaikki viestit käsitellään moduuleissa	R2	S1	T1
Käyttöliittymän Content proxy	NR2		
Tietokannan valikoiva suodatin	NR2		
Viestien päivitystaajuuden rajoitus RT API:ssa	NR2		
Viestit luodaan käyttäen QSharedPointereita.	NR3		
Viestinvälitysjono		S2	
Hierarkia		S3	
Järkeily			

Taulukko 6 Toisen skenaarion analyysin tarkennukset

Kohta	Järkeily
NR1	Laitteiston valmistajan ylläpitämän kirjaston ja ajurin käyttö järkevää.
NR2	Liian tiheät päivätykset suodatetaan korkeintaan 100 ms intervallille. Linjassa vaatimusten suhteen. Turvallinen ratkaisu.
NR3	Poistaa tarpeen tehdä turhia kopioita. Parantaa suorituskykyä, kun ainoa kopioitava muuttuja on osoitin. Parantaa koodin ylläpidettävyyttä ja luotettavuutta.
R1	Laitteiston valmistajan toteuttama kirjasto ja ajuri, jonka vialliseen toimintaan ei pysty vaikuttaan. Kirjastossa ja/tai ajurissa on ollut ongelmia ennenkin.
R2	Vaikka data ei sisällöltään muuttuisi, sitä ei suodateta. Kaikkien viestien käsittely ja välittäminen eteenpäin vain lisää järjestelmän kuormaa.
S1	Datan päivitysaika saadaan kirjattua ylös dataa tallentaviin moduuleihin.
S2	Jos tietovirta on suurempi kuin käsittelynopeus, FIFO-tyyppinen viestinvälitys johtaa siihen, että käyttöliittymälle tuleva data on vanhaa informaatiota.
S3	Hierarkian pituus vaikuttaa suorituskykyyn. Pituus lasketaan pisteillä erotettujen merkkijonojen määrällä, mikä on useimmissa tapauksissa kolme. Suorituskykyvaikutus suhteellisen pieni.
T1	Dataa ei voi suodateta tietolähdekerroksella, sillä jokin sovelluskerroksen moduuli voi vaatia muuttumatonta ja suodattamatonta tietoa riippumatta siitä, kuinka usein dataa vastaanotetaan. Datan luotettavuus(+) paranee, mutta heikentää suorituskykyä (-) lisäämällä turhaa prosessointia tietyissä tapauksissa.

Ensimmäinen riski, valmistajan kirjaston käyttö, osoittautui kiinnostavan kaksijakoiseksi: kirjasto nähtiin samaan aikana turvallisena ratkaisuna ja riskinä. Valmiin kirjaston käyttö nähtiin turvallisena ratkaisuna siinä mielessä, että ei ole perusteltua tehdä omaa kirjastototeutusta silloin, kun laitteiston valmistaja tarjoaa valmiin toteutuksen. Tämä ratkaisu osoittautui kuitenkin myös riskiksi, sillä CAN-kirjaston ja ajurin laatu on osoittautunut ohjelmiston kehityksen aikana kyseenalaiseksi. Ensinnä, kirjaston sanotaan toteuttavan CANpie-standardin mukaisen rajapinnan, mutta rajapinta ei vastaa standardia. Toisekseen kirjasto ei onnistu aina korjaamaan tiettyjä vakavia ohjelmistoperäisiä vikatilanteita, jotka voivat ilmentyä ilman selkeää pääteltävissä olevaa syytä.

Toiseksi riskiksi osoittautui se, että sisääntulevaa dataa ei suodateta. Tässä aliluvussa esiteltiin aikaisemmin datan eteneminen ohjelman sisällä ja se, miten jäsentäjä hylkää tuntemattomat. Jäsentäjän hyväksymät ja tulkkaamat viestit lähetetään joka kerta viestinvälittäjälle riippumatta siitä onko edeltävän viestin sisältö ollut identtinen vai ei.

Tämä siirtää viestien suodatusvastuun viestejä vastaanottaville moduuleille, jotka monessa tapauksessa eivät suodata viestejä tiettyjen reaaliaikarajapinnan ja tietokannan toiminnallisten vaatimusten takia (S1).

4.5 Riskit ja riskiteemat

Riskit nivoutuivat kolmeen eri teemaan: suorituskyky, rajapinnat ja käännösjärjestelmä. Suorituskykyyn liittyvät mahdolliset ongelmat liittyvät vahvasti ohjelmiston tulevaisuudennäkymiin ja niihin liittyviin tuntemattomiin tekijöihin. Ohjelmiston suorituskyky on kuitenkin tällä hetkellä hyvä ja laitteiston tehokkuudessa on vielä varaa, joten suorituskykyyn liittyvät riskit ovat lähinnä teoreettisia uhkia. Ohjelmaa on arkkitehtuuritasolla optimoitu siten, että suorituskyvylliset riskit rajoittuvat sisääntulevan datan määrään ja sen sisältöön.

Toiseksi riskiteemaksi nousi ohjelmiston rajapinnat ja siihen liittyvät lukemattomat tuntemattomat tekijät. Rajapinnoista ensimmäinen ja ongelmallisin on CAN-rajapinta, joka käsiteltiin aliluvussa 4.4.2. CAN rajapinnan lisäksi muissa laitteistorajapinnoissa on ollut ongelmia sen suhteen, että joko laitteisto ei ole toiminut täysin rajapintaspesifikaation mukaisesti tai rajapinta on muuttunut ohjelmiston kehityksen aikana. Riskialttiisiin rajapintoihin on laskettava myös pääohjelman tarjoama reaaliaikarajapinta ja huoltorajapinta. Riskit liittyvät useihin eri tiloihin, joita rajapinnan ohjelmat voivat saada ja niiden vaikutus ohjelman toimintaan. Rajapintojen muutokset voivat huolellisesta kapseloinnista huolimatta säteillä muutostarpeita myös muualle ohjelmaan, kuten hypoteettisena esimerkkinä huoltorajapintaan lisättävä ajoaikainen konfigurointituki, joka vaikuttaisi moniin ohjelman komponentteihin. Tämä on laskettava huomattavana riskinä ylläpidettävyyden, muunneltavuuden ja joissain tapauksissa luotettavuuden ominaisuuksiin.

Kolmanneksi riskiteemaksi nousi käännökseen ja varioimiseen liittyvät riskit, jotka ovat lähtöisin valitusta käännösjärjestelmästä ja Linux-alustasta. OpenEmbedded-alustan ja tuotteen kääntämisestä vastuussa olevien BitBake-reseptien ylläpitäminen vaatii erityisosaamista, mikä on jo itsessään laskettavissa projektinhallinnalliseksi riskiksi. BitBake-käännösjärjestelmä ei nykyisellä toteutuksella tue varioimista, joten mahdolliset alustan variointivaatimukset voivat olla mahdottomia toteuttaa. OpenEmbedded-alustassa piilee myös toinen ongelma: yhteensopivia asennettavia paketteja ei ole yhtä paljon kuin moderneissa Linuxin työasemadistribuutioissa. Tuotteen tulevissa versioissa voi olla tarve moderneille ominaisuuksille, joita tuetaan vain moderneissa Linux-distribuutioissa. OpenEmbedded-alustan käyttö voi rajata nämä ominaisuudet pois mahdottomina toteuttaa. Silläkin mahdollisuudella, että alustan muokkaus tulevaisuuden lukuisiin tarpeisiin olisi mahdollista, reseptien päivittämiseen vaaditaan BitBake-reseptien erityisosaamista omaava henkilö ja potentiaalisesti paljon aikaa.

5 ARVIOT JA JATKOKEHITYS

Tämä luku on kirjoitettu vuosi sen jälkeen kuin aikaisemmissa luvuissa käsitelty analyysi oli tehty. Siitä johtuen seuraavissa aliluvuissa esitetyt arkkitehtuurin jatkokehitys ja ATAM-menetelmän arvionti on toteutuman perusteella kirjoitettu. Lisäksi arvioidaan arkkitehtuuria analyysin tulosten valossa ja analyysin kustannustehokkuutta.

5.1 Ohjelmiston tila vuosi analyysin jälkeen

Ohjelmistoa on kehitetty aktiivisesti analyysin jälkeen ja vuoden kehittämisen jälkeen ohjelmistoon on tehty huomattavia muutoksia. Aliluvussa 4.5 käsiteltiin riskejä ja riskiteemoja, jotka arkkitehtuurista oli kartoitettu. Osa ohjelmiston kokemista muutoksista vastasi juuri näihin riskiteemoihin, kun taas osa riskeistä toteutui odottamattomalla tavalla. Riskit liittyivät suorituskykyyn, rajapintoihin ja käännösjärjestelmään. Näistä teemoista ainoastaan rajapinnat eivät liittyneet mihinkään muutoksiin tai toteutuneisiin riskeihin.

Suurin tuotteeseen tehty muutos liittyi Linux-käyttöjärjestelmään, jonka päällä ohjelmistoa suoritetaan. Tuotteen vanhempien varianttien käyttämä BitBake-käännösjärjestelmä nostettiin riskiksi siihen liittyvien lukuisien ongelmien ja haasteiden takia. Tämä järjestelmä korvattiin Linuxin modernimmalla, mutta räätälöidyllä työasemadistribuutiolla, johon ohjelmisto asennetaan erillisinä Debian-paketteina. Laitteiston viimeisimmässä versiossa on alkuperäiseen verrattuna huomattavasti enemmän resursseja, mistä johtuen vanhasta BitBake-käännösjärjestelmästä saatavat edut eivät ole enää tarpeellisia. Tästä syystä oli mahdollista alkaa käyttämään modernimpaa käyttöjärjestelmää.

Toinen riskiteema, joka liittyi mahdollisiin suorituskykyongelmiin, toteutui kahdella eri tavalla. Analyysin aikana oli odotettavissa, että ohjelmisto joutuu käsittelemään huomattavan määrän uusia viestejä aikaisempien lisäksi. Analyysissä tarkasteltiin suorituskykyyn liittyviä arkkitehtuuriratkaisuja ja mitään yksittäistä ongelmia aiheuttavaa paikkaa ei analyysin aikana löydetty. Uusien viestien lisäämisen jälkeen kävi ilmi, että tietokanta ei pysy vauhdissa mukana. Tallennettavia tietueita on hyvin suuri määrä, mistä johtuen tietokanta joutuu tekemään jatkuvasti tietokantatransaktioita ja siten koko järjestelmän suorituskyky heikkenee. Tietokantamoduulin sisäistä rakennetta ei oltu käsitelty analyysissä, sillä kukaan analyysissä läsnäolevista kehittäjistä ei tuntenut moduulin toteutusta. Tämä toteutunut riski aiheuttaa muutospainetta tietokantatoteutukseen ja siihen liittyviin arkkitehtuuriratkaisuihin.

Toinen suorituskykyyn vaikuttava merkittävä muutos on uusi QML-pohjainen (engl. Qt Meta Language, Qt Modeling Language) käyttöliittymä, joka korvasi edeltävän

widgetpohjaisen käyttöliittymän. QML on Qt-kehyksen tarjoama vaihtoehtoinen tapa tehdä käyttöliittymiä, missä muun muassa animaatioiden ja tehosteiden lisääminen on suhteellisen helppoa. Vaikka QML:n ytimeen oli tehty Qt:n versio 5:n myötä huomattavia suorituskykyparannuksia, suorituskyky ei ollut aivan sitä, mitä siltä voisi odottaa. Pääohjelman käyttöliittymän ollessa vielä laajempi kuin edeltävissä ohjelmiston versioissa renderoinnissa kuluva aika on paikoin silmin havaittavan pitkä. Suorituksen viiveet johtaa osalla käyttöliittymäsivuista alhaiseen ruudun kehys-nopeuteen (15 – 30 fps), minkä huomaa käyttöliittymäelementtien animaatioissa. Tähän vaikuttaa myös paikoin tietokannan aiheuttama suuri kuorma, jolloin alhaisemmalla prioriteetilla CPU-aikaa saava käyttöliittymä joutuu luovuttamaan omaa aikaansa.

Muilta osin arkkitehtuuri pysyi pääosin ennallaan. Ohjelmiston vaatimukset eivät ole muuttuneet kovin paljon sen ensimmäisestä versiosta lähtien, joten alkuperäinen arkkutehtuuri palvelee edelleen hyvin tarkoitustaan. Toteutuneet riskit ovat aiheutuneet enimmäkseen siitä, että ohjelmiston vaatimukset ovat ajan myötä muuttuneet, jolloin alkuperäiset arkkitehtuuriratkaisut eivät enää tukeneet ohjelmistoon tehtyjä muutoksia. Ohjelmiston ikään nähden toteutuneiden riskien määrä on suhteellisen pieni, mutta odotettavissa oleva.

5.2 Arkkitehtuurin sopivuus vaadittuun tehtävään

Vaikka ATAM-menetelmän tarkoitus ei olekaan antaa arviota siitä kuinka hyvä arkkitehtuuri on, menetelmän tulosten perusteella voi päätellä soveltuuko arkkitehtuuri sille annettuihin tehtäviin. Arkkitehtuurin soveltuvuutta voi arvioida niin riskien pohjalta kuin sen, miten hyvin arkkitehtuuri toimii tärkeimpien skenaarioiden tapauksessa.

Skenaarioita laadittaessa suurin osa skenaarioista oli hyvin lähellä todellisia käyttötapauksia ja ohjelman kehittäjillä oli vaikeuksia keksiä skenaarioita, joihin arkkitehtuuri ei olisi jo käytännön tasolla todettu toimivaksi. Arkkitehtuurin rajoja koettelevia skenaarioita onnistuttiin laatimaan vasta, kun asiakas osallistui analyysikokoukseen. Nämä uudet skenaariot olivat pääosin tutkivia skenaarioita ja mahdollisia tulevaisuuden tarpeista kumpuavia laajennettavuuteen takertuvia skenaarioita. Mikäli arkkitehtuuri olisi kovin huono, se ei kestäisi tarkastelua edes tehtäviltään tärkeimpien skenaarioiden tapauksessa.

Analyysin myötä järjestelmästä löytyi riskejä, joista suurin osa oli jo ennalta tiedossa. Riskien teemoissa oli paljon eroja ja osaan riskeistä on vaikea reagoida, kuten olemassa olevien rajapintojen muuttaminen tai Qt-kehyksessä piilevät riskit. Vuoden aikana ilmenneissä riskeissä ei kuitenkaan ole mitään, mihin ei pystyisi kohtuullisella työllä vastaamaan. Muiden riskien realisoituminen on pääosin riippuvainen siitä, minkälaisia muutoksia ohjelmisto tulevaisuudessa tulee kokemaan.

Analyysissä julki tulleet asiakkaan liiketoimintanäkökulmat olivat suurelta osin niitä, joita ohjelmistoa kehitettäessä oli tullut ilmi. Tämän lisäksi kyseinen arkkitehtuuri on versionumeroltaan neljäs. Sitä kehitettäessä on otettu huomioon kolmannen version

puutteet ja sen kokema tehtävänkuva. Analyysin perusteella arkkitehtuuri vastaa hyvin nykyisiin haasteisiin ja osa asiakkaan esittämistä tulevaisuuden mahdollisista muutoksista ja laajennoksista on helppo integroida nykyiseen arkkitehtuuriin. Asiakkaan avulla laadittujen tutkivien skenaarioiden myötä tuli ilmi skenaarioita, jonka perusteella osa ohjelmiston kokonaisrakenteesta voi kokea isompiakin muutoksia, mutta niissäkin tapauksissa arkkitehtuurin ydin ja suurin osa moduuleista ei vaatisi mitään muutoksia.

Arkkitehtuuria suunniteltaessa kävi ilmi, että odotettu elinkaari olisi pitkä ja saman ydinarkkitehtuurin perusteella tehtäisiin useampi tuote. Usean vuoden kehityksen aikana ja tämän analyysin perusteella ei ole ilmentynyt syitä, miksi arkkitehtuuri ei vastaisi hyvin nykyisiin tarpeisiin. Laadittujen skenaarioiden ja asiakkaan liiketoiminta-näkökulmien perusteella tärkeimmät vaatimukset on katettu hyvin, eikä ole odotettavissa, että jo löydettyihin riskeihin vastaaminen vaatisi mittavia muutoksia. On todennäköistä, että ainoastaan tulevaisuudessa lisääntyvät ja muuttuvat vaatimukset voivat johtaa siihen, että arkkitehtuuri täytyy vaihtaa tai muokata.

5.3 ATAM-menetelmän arviointi

SEI on antanut arvionsa ATAM-menetelmästä sen perusteella, miten se on toiminut vuosien varrella tehdyissä analyyseissä. Hyödyt ovat ilmeiset, mikäli projektin kokoluokka on riittävän iso ja analyysiä tekevät henkilöt tietävät mitä tekevät. Kiinnostavinta on kuitenkin se, miten menetelmä suoriutuu pienemmässä mittakaavassa, sillä hyödyt ja haitat skaalautuvat eri tavalla.

Aliluvussa 2.4 on käsitelty menetelmän hyötyjä ja haittoja niiltä osin, miten menetelmän kehittäjät ovat ATAM-menetelmää arvioineet. Ilmeiset haitat ovat menetelmän suoritukseen kuluva aika ja kustannukset. Projektin ollessa pieni analyysin kustannukset voivat kasvaa huomattavasti suuremmiksi kuin analyysistä koituvat säästöt. Hyvin pienessä projektissa jopa modifioitu ja karsittu versiokin saattaa olla aivan liian raskas sovellettavan projektin kompleksisuuteen ja mittakaavaan nähden.

Vaikka ATAM-menetelmä ei käytä yksittäisten henkilöiden aikaa erityisen paljon, kalenteriaikaa aloituksesta lopetukseen voi kulua hyvin paljon. Mikäli analyysi haluttaisiin tehtävän heti arkkitehtuurisuunnittelun jälkeen, mutta ennen implementointia, kaikki analyysiin kuluva aika lykkää implementoinnin aloitusajankohtaa. Suurissa ja pitkäkestoisissa projekteissa tällä seikalla ei ole erityisesti merkitystä, mikäli analyysi lyhentää implementointiaikaa merkittävästi, mutta lyhyissä ja kiireellisissä projekteissa projektin päättymisajankohta saattaa lykkääntyä. Merkittävimpiä tekijöitä tässä on se, kuinka optimaalisesti tärkeimpien osallistujien aikataulut sopivat nopeatahtiseen analyysiin. Mikäli nopean aikataulun järjestäminen koituu mahdottomaksi, on arvioitava tilannekohtaisesti kannattaako analyysi tehdä ATAM-menetelmän mukaisesti.

ATAM-menetelmää sovellettaessa käytäntöön on huomattavissa hyvin nopeasti, että itse menetelmän kirjanopillinen tapa toimii paremmin isompien ryhmien

hallitsemisessa. Menetelmän teoriassa on esitelty lukuisia määriä erilaisia rooleja ja toimenpiteitä, joita evaluointitiimin pitää suorittaa, mutta jotka vaativat myös oman vaivannäkönsä. Menetelmän vaiheistus useampaan vaiheeseen on selkeästi suunniteltu sitä varten, että ison osallistujamäärän tapauksessa organisoiminen on selkeää ja luontevaa. Pienellä osallistujamäärällä menetelmän läpikäynti prosessinmukaisen kankeasti tuottaa tarpeettoman paljon ylimääräistä työtä. Jos esimerkiksi evaluoinnissa olisi muista sidosryhmistä vain yksi henkilö, jälkimmäisen skenaarioaivoriihen äänestys (aliluku 2.2.4, toimenpide 7) ATAM-teorian mukaisesti menettää merkityksensä.

Menetelmän tärkein tulos on arkkitehtuurin riskianalyysi, joka antaa tärkeää tietoa arkkitehtuurista. Mikäli arkkitehtuurissa ilmenee pahoja puutteita, johon tarvitaan uusia arkkitehtuuriratkaisuja, nämä ratkaisut ovat luonnollisesti potentiaalisia riskinlähteitä. Tämä johtaa siihen, että arkkitehtuuri saattaa kaivata uudelleenevaluointia, mikä voisi johtaa usean kalenteripäivän viiveeseen. Pienen projektin tapauksessa on kuitenkin oletettavaa, että arkkitehtuuri on yksinkertaisempi ja oleellisimmat evaluointityövaiheet on nopeasti käyty läpi ja siten evaluointitapahtumassa saattaa jäädä ylimääräistä aikaa. Vaikka menetelmän ei ole tarkoitus tarjota ratkaisuja löydettyihin ongelmiin, osallistujien ajan kuluttaminen epäolennaisiin skenaarioihin ei ole tuotantotavoitteellisesti järkevää. Tämä mahdollistaisi sen, että koko evaluointiin kokoontuneella arkkitehtuuriosaamisella voisi suunnitella ratkaisut pahimpiin riskeihin ja evaluoida uudelleen ne aikaisemmin määriteltyjä skenaarioita vastaan.

SEI:n määrittelemä ATAM-menetelmä sopii sellaisenaan isoihin projekteihin, koska menetelmän hyödyistä on näyttöä. Pienen projektin tapauksessa menetelmä on sellaisenaan kuitenkin liian raskas ja kankea. Jotta menetelmän karsittu versio toisi projektille hyötyjä, on menetelmää suoritettaessa joustettava resurssi- ja aikataulupaineiden alla. Tämä vaatii evaluoijalta kokemusta mukautua tilanteiden ja tarpeiden mukaan. Mikäli menetelmää muokkaa paljon, sopii kysyä onko menetelmä enään ATAM, vai onko se jotain muuta. Pienen projektin tapauksessa on arvoitava kannattaako evaluointi suorittaa ATAM-menetelmän jollain modifikaatiolla, vai kannattaisiko käyttää jotain kevyempää evaluointimenetelmää.

5.4 Hyödyn suhde kustannuksiin

Teorian mukaan arkkitehtuurianalyysistä saatavat hyödyt ovat suuremmat, kun analyysi suoritetaan ennen implementaation aloittamista. ATAM-teoriassa esitetään myös, että implementaation jälkeen tehdystä analyysistä on hyötyä, mutta saadut tulokset ja hyödyt ovat silloin erilaisia. Sama pätee tähän analyysiin. Jälkikäteen tehdystä analyysistä saadaan hyvää tietoa tulevia jatkokehityshankkeita varten, ja tämä on hyödyllisintä silloin, kun ohjelmiston kehittäjät ovat vaihtuneet. Nämä hyödyt ovat jääneet kuitenkin minimaalisiksi, sillä ohjelmistoa analysoivat samat henkilöt kuin ne jotka olivat tekemässä arkkitehtuuria. Tästä johtuen analyysistä saatavan uuden informaation määrä pysyy pienenä.

Analyysiin kulutettu aika on hyvin pieni, sillä rahallinen panostus jäi pelkästään analyysikokouksiin kulutettuihin miestyöpäiviin. Aliluvun 4.2.1 mukaan määrärahoja kului analyysiin yhteensä seitsemän miestyöpäivän edestä. Esivalmisteluihin ja jälkianalyyseihin kulutettua aikaa ei pystytä suoraan laskemaan, sillä ne ovat käytetty pääosin tämän diplomityön kirjallisen osuuden tekemisessä. Kulutettu aika ja siitä tulevat kustannukset ovat kokonaisuudessaan melko pieniä, minkä voidaan päätellä johtuvan projektin pienestä koosta ja osallistujien hyvästä arkkitehtuurituntemuksesta.

Selkeitä hyötyjä projektitiimin kannalta oli asiakkaan liiketoimintanäkökulmien tarkentuminen, sekä tuotekehityksen tulevaisuudennäkymät, jotka tulivat julki jälkimmäisen analyysin aikana. Asiakkaan kannalta hyödyllistä oli nähdä rajoitteita, joita arkkitehtuuri asetti tulevaisuudennäkymiin pohjautuvien tutkivien skenaarioiden tapauksissa. Tämä on erityisesti hyödyllistä siitä johtuen, että asiakas on vastuussa tämän ohjelmiston tulevaisuudesta ja sen kehitykseen kuluvista kustannuksista. Rajoitusten ollessa selvillä tulevien jatkokehityshankkeiden tapauksessa tuotetta ei aktiivisesti kehitetä suuntaan, minkä hyöty-kustannussuhde on heikko.

Realisoituneita hyötyjä ei ole mahdollista laskea matemaattisesti, sillä ohjelmiston vaihtoehtoisia kehityspolkuja ei pystytä erikseen tarkastelemaan. Ainoa analyysiin liittyvä konkreettinen toimenpide oli Linux-alustan uusiminen, minkä aikataulua analyysin tulokset vauhdittivat. Muilta osin analyysistä saatavat hyödyt saattavat ilmentyä pitkällä aikajänteellä. Henkiseen pääomaan sitoutuneita hyötyjä on mahdoton arvioida. Analyysin kustannukset ovat olleet kokonaisuudessaan suhteellisen pienet teorian opettelua ja diplomityön osuutta lukuunottamatta, joten kustannukset on katettu jo pienienkin tulevien kustannussäästöjen myötä. Vaikka hyödyt jäisivät kustannusten varjoon, menetetyn pääoman määrä jää hyvin minimaaliseksi.

6 YHTEENVETO

Tämän diplomityön puitteissa on pyritty soveltamaan ATAM-menetelmää pienprojektiin ja arvioimaan sovelluksen toimivuutta. ATAM-menetelmän kehittänyt SEI esittää menetelmän rakenteen ja toiminnan sen täydessä mittakaavassa, jolloin suurille projekteille suunnitellusta analyysistä johtuvat kustannukset ovat pienprojektien tapauksessa todennäköisesti tarpeettoman suuret. SEI esittää ATAM-teoriassaan myös hyvin keveän version, mikä antaa osviittaa siitä, että menetelmä taipuu myös pienempään mittakaavaan. Kevennetyn ATAM:n on esitetty toimivan tiimillä, joka tuntee teorian jo ennalta, jolloin analyysi etenisi vauhdikkaammin. Tämän muunnelman esittely on kirjallisuudessa kuitenkin hyvin marginaalista, jolloin menetelmän käyttäjät joutuvat itse soveltamaan tilannekohtaiset muunnelmansa.

Diplomityössä on sovellettu ATAM-menetelmää pienehköön projektiin, jonka aktiivinen kehitystiimi on kooltaan keskimäärin kaksihenkinen. Projektin tuotteesta on varioitu useampi tuote, mikä tarkoittaa sitä, että arkkitehtuurianalyysi on tehty implementaation jälkeen. Tietäen, että tuotetta tullaan varioimaan jatkossakin, analyysin skenaarioiksi laadittiin paljon tulevaisuudennäkymiin perustuvia skenaarioita. Tällä pyrittiin saamaan uutta tietoa arkkitehtuurista ja samalla välttämään tilanne, että skenaarioihin käytetyt voimavarat toisivat julki vain jo tiedettyjä asioita.

ATAM-menetelmän kevennyksessä haettiin suurimmat kustannussäästöt analysointitiimin koosta. Analysointitiimin jäsenmäärän karsimisessa pyrittiin keskittymään vain tärkeimpien tiimiläisten säilyttämiseen. Jokaisen jäsenen kohdalla voidaan arvioida kustannustehokkuutta heidän roolin ja jaettujen roolien päällekkäisyyksien perusteella. Analyysin tärkeimmiksi jäseniksi muodostui arkkitehti, projektipäällikkö, asiakas ja evaluoinninjohtaja.

Toisena kustannussäästökohteena oli tehtävät työvaiheet. Ison projektin ja ison evaluointitiimin tapauksessa kaikkien työvaiheiden voi olettaa olevan tärkeitä ja tarpeellisia. Tuskin ATAM-teoriassa olisi kaikkia työvaiheita määritelty, jos niiden tarpeellisuus ei olisi käynyt ilmi SEI:n kehittäessä ATAM-menetelmää. Pienemmän projektin puitteissa on kuitenkin hallittavissa huomattavasti ketterämpi lähestymistapa analysointiin. Kun tiimin koko ja projektin pituus on helpohkosti hahmotettavissa, on mahdollista tehdä jälleen kustannustehokkuusarvioita eri työvaiheille. Tämän analyysin puitteissa oli mahdollista supistaa ATAM-teorian ja arkkitehtuurin esittely minimiin, sillä arkkitehtuuri oli kaikille ennestään tuttu ja analyysitiimi oli helposti hallittavissa pienelläkin teoriaperehdytyksellä.

Koska ATAM-menetelmää sovellettiin valmiiseen tuotteeseen, tavallisimpia skenaarioita ei nähty kovin hyödyllisiksi analyysin kohteiksi. Menetelmän anti olisi jäänyt laihaksi, mikäli analyysissä olisi visusti pysytty hyvin testatuissa ja käytäntöön

sovelletuissa käyttötapauksissa. Tästä johtuen analyysissä laadittiin myös hypoteettisia skenaarioita, joiden toteutumiseen voi suhtautua skeptisesti. Nämä skenaariot laadittiin siltä pohjalta, että arkkitehtuuriin tehtiin jonkin uuden toiminnallisuuden vaatima muutos ja tarkasteltiin sitä kuinka muutettu arkkitehtuuri toimii kyseisissä skenaariossa. Loppujen lopuksi analyysistä muodostui osittain foorumi, missä sekä analysoitiin tuotetta, että keskusteltiin tuotteen tulevaisuudesta ja siihen liittyvistä muutoksista. Analyysissä ei siis pysytty tiukasti ATAM-menetelmän raameissa, vaan siitä poikettiin niiltä osin, missä menetelmän kurinalainen noudattaminen ei vaikuttanut antavan lisäarvoa.

Projekti, missä ATAM-menetelmää käytettiin, ei ollut ideaalisin mahdollinen tuomaan esille menetelmän parhaita hyötyjä. Menetelmän käytäntöönpano antoi kuitenkin osviittaa siitä, miten menetelmä toimii ja mitä arkkitehtuurin analysoimisesta on mahdollista saada irti. Kustannuksiltaan koko prosessi ei ollut kovin mittava ja harjoituksen myötä kustannustehokkuuden voi olettaa parantuvan. Jotta ATAM-muunnelman hyödyllisyydestä voisi antaa luotettavan arvion, sitä pitäisi soveltaa useampaan pienprojektiin. Siitäkin huolimatta, tämän yhden, hieman epäideaalisen sovelluskohteen perusteella voi suositella ATAM-menetelmän soveltamista myös pieniin projekteihin.

LÄHTEET

[1] [Clements 2002] Clements, P., Kazman, R. & Klein, M. 2002. Evaluating Software Architectures: Methods and Case Studies. Addison-Wesley. 323 s.

[2] [Bass 2003] Bass, L., Clements, P. & Kazman, R. 2003. Software Architecture in Practice. 2nd edition. Addison-Wesley. 528 s.

[3] [Bass 2012] Bass, L., Clements, P. & Kazman, R. 2012. Software Architecture in Practice. 3rd edition. Addison-Wesley. 640 s.

[4] [Koskimies 2005] Koskimies, K. & Mikkonen, T. 2005. Ohjelmistoarkkitehtuurit. Talentum oyj. 250 s.

[5] [Kazman] Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H. & Carriere, J. Kesäkuu 1998. The Architecture Tradeoff Analysis Method. CMU/SEI-98-TR-008. 40 s. [viitattu 2.11.2012]. Saatavissa: http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA350761

[6] [Abowd 97] Abowd, G., Bass, L., Clements, P., Kazman, R., Northrop, L. & Zaremski, A. Tammikuu 1997. Recommended Best Industrial Practice for Software Architecture Evaluation. CMU/SEI-96-TR-025. 34 s. [viitattu 7.11.2012]. Saatavissa: http://www.sei.cmu.edu/reports/96tr025.pdf

[7] Quality Attribute Workshop, [viitattu 7.11.2012]. Saatavissa: http://www.sei.cmu.edu/architecture/tools/establish/qaw.cfm

[8] Ionita, M.T., Hammer, D.K., Obbink, H. Scenario-Based Software Architecture Evaluation Methods: An Overview. [viitattu 9.11.2012]. Saatavissa: http://pdf.aminer.org/000/359/847/prioritizing_scenario_evolution.pdf

[9] Nord, R., Barbacci, M., Clements, P., Kazman, R., Klein, M., O'Brien, L. & Tomayko, J. 2003. Integrating the Architecture Tradeoff Analysis Method (ATAM) with the Cost Benefit Analysis Method (CBAM). CMU/SEI-2003-TN-038. Saatavissa: http://www.sei.cmu.edu/reports/03tn038.pdf